

*A Transformation-based Implementation for CLP with Qualification and Proximity **

Preliminary Version (Technical Report SIC-4-10)

R. CABALLERO, M. RODRÍGUEZ-ARTALEJO and C. A. ROMERO-DÍAZ

Departamento de Sistemas Informáticos y Computación, Universidad Complutense

Facultad de Informática, 28040 Madrid, Spain

(e-mail: {rafa,mario}@sip.ucm.es, cromdia@fdi.ucm.es)

Abstract

Uncertainty in logic programming has been widely investigated in the last decades, leading to multiple extensions of the classical LP paradigm. However, few of these are designed as extensions of the well-established and powerful CLP scheme for Constraint Logic Programming. In a previous work we have proposed the SQCLP (*proximity-based qualified constraint logic programming*) scheme as a quite expressive extension of CLP with support for qualification values and proximity relations as generalizations of uncertainty values and similarity relations, respectively. In this paper we provide a transformation technique for transforming SQCLP programs and goals into semantically equivalent CLP programs and goals, and a practical Prolog-based implementation of some particularly useful instances of the SQCLP scheme. We also illustrate, by showing some simple—and working—examples, how the prototype can be effectively used as a tool for solving problems where qualification values and proximity relations play a key role. Intended use of SQCLP includes flexible information retrieval applications.

KEYWORDS: Constraint Logic Programming, Program Transformation, Qualification Domains and Values, Similarity and Proximity Relations, Flexible Information Retrieval.

1 Introduction

Many extensions of LP (*logic programming*) to deal with uncertain knowledge and uncertainty have been proposed in the last decades. These extensions have been proposed from different and somewhat unrelated perspectives, leading to multiple approaches in the way of using uncertain knowledge and understanding uncertainty.

A recent work by us (Rodríguez-Artalejo and Romero-Díaz 2010a) focuses on the declarative semantics of a new proposal for an extension of the CLP scheme supporting qualification values and proximity relations. More specifically, this work defines a new generic scheme SQCLP (*proximity-based qualified constraint logic programming*) whose instances SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) are parameterized by a proximity

* This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), PROMETIDOS-CM (S2009TIC-1465) and GPD-UCM (UCM-BSCH-GR58/08-910502).

relation \mathcal{S} , a qualification domain \mathcal{D} and a constraint domain \mathcal{C} . The current paper is intended as a continuation of (Rodríguez-Artalejo and Romero-Díaz 2010a) with the aim of providing a semantically correct program transformation technique that allows us to implement a sound and complete implementation of some useful instances of SQCLP on top of existing CLP systems like *SICStus Prolog* or *SWI-Prolog*. In the introductory section of (Rodríguez-Artalejo and Romero-Díaz 2010a) we have already summarized some related approaches of SQCLP with a special emphasis on their declarative semantics and their main semantic differences with SQCLP. In the next paragraphs we present a similar overview but, this time, putting the emphasis on the goal resolution procedures and system implementation techniques, when available.

Within the extensions of LP using annotations in program clauses we can find the seminal proposal of *quantitative logic programming* by (van Emden 1986) that inspired later works such as the GAP (*generalized annotated programs*) framework by (Kifer and Subrahmanian 1992) and the QLP (*qualified logic programming*) scheme by us (Rodríguez-Artalejo and Romero-Díaz 2008). In the proposal of van Emden, one can find a primitive goal solving procedure based in and/or trees (these are similar to the alpha-beta trees used in game theory), used to prune the search space when proving some specific ground atom for some certainty value in the real interval $[0, 1]$. In the case of GAP, the goal solving procedure uses constrained SLD resolution in conjunction with a—costly—computation of so-called *reductants* between variants of program clauses. In contrast, QLP goal solving uses a more efficient resolution procedure called SLD(\mathcal{D}) resolution, implemented by means of real domain constraints, used to compute the qualification value of the head atom based on the attenuation factor of the program clause and the previously computed qualification values of the body atoms. Admittedly, the gain in efficiency of SLD(\mathcal{D}) w.r.t. GAP’s goal solving procedure is possible because QLP focuses on a more specialized class of annotated programs. While in all these three approaches there are some results of soundness and completeness, the results for the QLP scheme are the stronger ones (again, thanks to its also more focused scope w.r.t. GAP).

From a different viewpoint, extensions of LP supporting uncertainty can be roughly classified into two major lines: approaches based in fuzzy logic (Zadeh 1965; Hájek 1998) and approaches based in similarity relations. Historically, Fuzzy LP languages were motivated by expert knowledge representation applications. Early Fuzzy LP languages implementing the resolution principle introduced in (Lee 1972) include Prolog-Elf (Ishizuka and Kanai 1985), Fril Prolog (Baldwin et al. 1995) and F-Prolog (Li and Liu 1990). More recent approaches such as the Fuzzy LP languages in (Vojtáš 2001; Guadarrama et al. 2004) and Multi-Adjoint LP (MALP for short) in the sense of (Medina et al. 2001a) use clause annotations and a fuzzy interpretation of the connectives and aggregation operators occurring in program clauses and goals. The Fuzzy Prolog system proposed in (Guadarrama et al. 2004) is implemented by means of real constraints on top of a CLP(\mathcal{R}) system, using a syntactic expansion of the source code during the Prolog compilation. A complete procedural semantics for MALP using reductants has been presented in (Medina

et al. 2001b). A method for translating a MALP like program into standard Prolog has been described in (Julián et al. 2009).

The second line of research mentioned in the previous paragraph was motivated by applications in the field of flexible query answering. Classical LP is extended to Similarity-based LP (SLP for short), leading to languages which keep the classical syntax of LP clauses but use a similarity relation over a set of symbols S to allow “flexible” unification of syntactically different symbols with a certain approximation degree. Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as fuzzy relations represented by mappings $\mathcal{S} : S \times S \rightarrow [0, 1]$ which satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. Resolution with flexible unification can be used as a sound and complete goal solving procedure for SLP languages as shown e.g. in (Sessa 2002). SLP languages include *Likelog* (Arcelli and Formato 1999; Arcelli Fontana 2002) and more recently *SiLog* (Loia et al. 2004), which has been implemented by means of an extended Prolog interpreter and proposed as a useful tool for web knowledge discovery.

In the last years, the SLP approach has been extended in various ways. The SQLP (*similarity-based qualified logic programming*) scheme proposed in (Caballero et al. 2008) extended SLP by allowing program clause annotations in QLP style and generalizing similarity relations to mappings $\mathcal{S} : S \times S \rightarrow D$ taking values in a qualification domain not necessarily identical to the real interval $[0, 1]$. As implementation technique for SQLP, (Caballero et al. 2008) proposed a semantically correct program transformation into QLP, whose goal solving procedure has been described above. Other related works on transformation-based implementations of SLP languages include (Sessa 2001; Medina et al. 2004). More recently, the SLP approach has been generalized to work with *proximity relations* in the sense of (Dubois and Prade 1980) represented by mappings $\mathcal{S} : S \times S \rightarrow [0, 1]$ which satisfy reflexivity and symmetry axioms but do not always satisfy transitivity. SLP like languages using proximity relations include Bousi~Prolog (Julián-Iranzo and Rubio-Manzano 2009a) and the SQCLP scheme (Rodríguez-Artalejo and Romero-Díaz 2010a). Two prototype implementations of Bousi~Prolog are available: a low-level implementation (Julián-Iranzo and Rubio-Manzano 2009b) based on an adaptation of the classical WAM (called *Similarity WAM*) implemented in JAVA and able to execute a Prolog program in the context of a similarity relation defined on the first order alphabet induced by that program; and a high-level implementation (Julián-Iranzo et al. 2009) done on top of *SWI-Prolog* by means of a program transformation from Bousi~Prolog programs into a so-called *Translated BPL code* than can be executed according to the weak SLD resolution principle by a meta-interpreter.

Let us now refer to approaches related to constraint solving and CLP. An analogy of proximity relations in the context of partial constraint satisfaction can be found in (Freuder and Wallace 1992), where several metrics are proposed to measure the proximity between the solution sets of two different constraint satisfaction problems. Moreover, some extensions of LP supporting uncertain reasoning use constraint solving as implementation technique, as discussed in the previous paragraphs. However, we are only aware of three approaches which have been conceived

as extensions of the classical CLP scheme proposed for the first time in (Jaffar and Lassez 1987). These three approaches are: (Riezler 1998) that extends the formulation of CLP by (Höhfeld and Smolka 1988) with quantitative LP in the sense of (van Emden 1986) and adapts van Emden’s idea of and/or trees to obtain a goal resolution procedure; (Bistarelli et al. 2001) that proposes a semiring-based approach to CLP, where constraints are solved in a soft way with levels of consistency represented by values of the semiring, and is implemented with `clp(FD,S)` for a particular class of semirings which enable to use local consistency algorithms, as described in (Georget and Codognet 1998); and the SQCLP scheme proposed in our previous work (Rodríguez-Artalejo and Romero-Díaz 2010a), which was designed as a common extension of SQLP and CLP.

As we have already said at the beginning of this introduction, this paper deals with transformation-based implementations of the SQCLP scheme. Our main results include: a) a transformation technique for transforming SQCLP programs into semantically equivalent CLP programs via two specific program transformations named elim_S and elim_D ; and b) and a practical Prolog-based implementation which relies on the aforementioned program transformations and supports several useful SQCLP instances. As far as we know, no previous work has dealt with the implementation of extended LP languages for uncertain reasoning which are able to support clause annotations, proximity relations and CLP style programming. In particular, our previous paper (Caballero et al. 2008) only presented a transformation analogous to elim_S for a programming scheme less expressive than SQCLP, which supported neither non-transitive proximity relations nor CLP programming. Moreover, the transformation-based implementation reported in (Caballero et al. 2008) was not implemented in a system.

The reader is assumed to be familiar with the semantic foundations of LP (Lloyd 1987; Apt 1990) and CLP (Jaffar and Lassez 1987; Jaffar et al. 1998). The rest of the paper is structured as follows: Section 2 presents a brief overview of the semantics of the SQCLP scheme, focusing on the essential notions needed to understand the following sections and concluding with an abstract discussion of goal solving procedures for SQCLP. Section 3 briefly discusses two specializations of SQCLP, namely QCLP and CLP, which are used as the targets of the program transformations elim_S and elim_D , respectively. Section 4 presents these two program transformations along with mathematical results which prove their semantic correctness, relying on the declarative semantics of the SQCLP, QCLP and CLP schemes. Section 5 presents a Prolog-based prototype system which relies on the transformations proposed in the previous section and implements several useful SQCLP instances. Finally, Section 6 summarizes conclusions and points to some lines of planned future research.

2 The Scheme SQCLP and its Declarative Semantics

We present in this section a short overview of the declarative semantics of the SQCLP scheme originally presented in (Rodríguez-Artalejo and Romero-Díaz 2010a), focusing on the essential notions needed to understand the following sections. In-

interested readers are referred to (Rodríguez-Artalejo and Romero-Díaz 2010a) and its extended version (Rodríguez-Artalejo and Romero-Díaz 2010b) for a full-fledged exposition of SQCLP semantics and a discussion of various extended LP languages for uncertain reasoning which can be obtained as specializations and instances of SQCLP. Some technical notions and results from (Rodríguez-Artalejo and Romero-Díaz 2010b) will be cited along this paper when needed to support mathematical proofs.

Constraint domains \mathcal{C} , sets of constraints Π and their solutions, as well as terms, atoms and substitutions over a given \mathcal{C} are well known notions underlying the CLP scheme. The reader is referred (Rodríguez-Artalejo and Romero-Díaz 2010b) for a relational formalization of constraint domains and some examples, including the real constraint domain \mathcal{R} . We assume the following classification of atomic \mathcal{C} -constraints: defined atomic constraints $p(\bar{t}_n)$, where p is a program-defined predicate symbol; primitive constraints $r(\bar{t}_n)$ where r is a \mathcal{C} -specific primitive predicate symbol; and equations $t == s$.

We use $\text{Con}_{\mathcal{C}}$ as a notation for the set of all \mathcal{C} -constraints and κ as a notation for an atomic primitive constraint. Constraints are interpreted by means of \mathcal{C} -valuations $\eta \in \text{Val}_{\mathcal{C}}$, which are ground substitutions. The set $\text{Sol}_{\mathcal{C}}(\Pi)$ of solutions of $\Pi \subseteq \text{Con}_{\mathcal{C}}$ includes all the valuations η such that $\Pi\eta$ is true when interpreted in \mathcal{C} . $\Pi \subseteq \text{Con}_{\mathcal{C}}$ is called *satisfiable* if $\text{Sol}_{\mathcal{C}}(\Pi) \neq \emptyset$ and *unsatisfiable* otherwise. $\pi \in \text{Con}_{\mathcal{C}}$ is *entailed* by $\Pi \subseteq \text{Con}_{\mathcal{C}}$ (noted $\Pi \models_{\mathcal{C}} \pi$) iff $\text{Sol}_{\mathcal{C}}(\Pi) \subseteq \text{Sol}_{\mathcal{C}}(\pi)$.

Qualification domains were first introduced in (Rodríguez-Artalejo and Romero-Díaz 2008) with the aim of providing elements, called qualification values, which can be attached to computed answers. They are defined as structures $\mathcal{D} = \langle D, \leq, \mathbf{b}, \mathbf{t}, \circ \rangle$ verifying the following requirements:

1. $\langle D, \leq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points \mathbf{b} (called *infimum* or *bottom* element) and \mathbf{t} (called *maximum* or *top* element) w.r.t. the partial ordering \leq (called *qualification ordering*). For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e , and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e . We also write $d \triangleleft e$ as abbreviation for $d \leq e \wedge d \neq e$.
2. $\circ : D \times D \rightarrow D$, called *attenuation operation*, verifies the following axioms:
 - (a) \circ is associative, commutative and monotonic w.r.t. \leq .
 - (b) $\forall d \in D : d \circ \mathbf{t} = d$ and $d \circ \mathbf{b} = \mathbf{b}$.
 - (c) $\forall d, e \in D : d \circ e \leq e$ and even $\mathbf{b} \neq d \circ e \leq e$ if $d, e \in D \setminus \{\mathbf{b}\}$.
 - (d) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = (d \circ e_1) \sqcap (d \circ e_2)$.

For any $S = \{e_1, e_2, \dots, e_n\} \subseteq D$, the *glb* (also called *infimum* of S) exists and can be computed as $\sqcap S = e_1 \sqcap e_2 \sqcap \dots \sqcap e_n$ (which reduces to \mathbf{t} in the case $n = 0$). The dual claim concerning *lubs* is also true. As an easy consequence of the axioms, one gets the identity $d \circ \sqcap S = \sqcap \{d \circ e \mid e \in S\}$.

Technical details, explanations and examples can be found in (Rodríguez-Artalejo and Romero-Díaz 2010b), including: the qualification domain \mathcal{B} of classical boolean values, the qualification domain \mathcal{U} of uncertainty values, the qualification domain \mathcal{W} of weight values, and other qualification domains built from these by means of

the strict cartesian product operation \otimes . The following definition is borrowed from (Rodríguez-Artalejo and Romero-Díaz 2010a):

Definition 2.1 (Expressing \mathcal{D} in \mathcal{C})

A qualification domain \mathcal{D} is expressible in a constraint domain \mathcal{C} if there is an injective embedding mapping $\iota : D \setminus \{\mathbf{b}\} \rightarrow C$ and moreover:

1. There is a \mathcal{C} -constraint $\mathbf{qVal}(X)$ such that $\text{Sol}_{\mathcal{C}}(\mathbf{qVal}(X))$ is the set of all $\eta \in \text{Val}_{\mathcal{C}}$ verifying $\eta(X) \in \text{ran}(\iota)$.
2. There is a \mathcal{C} -constraint $\mathbf{qBound}(X, Y, Z)$ encoding “ $x \trianglelefteq y \circ z$ ” in the following sense: any $\eta \in \text{Val}_{\mathcal{C}}$ such that $\eta(X) = \iota(x)$, $\eta(Y) = \iota(y)$ and $\eta(Z) = \iota(z)$ verifies $\eta \in \text{Sol}_{\mathcal{C}}(\mathbf{qBound}(X, Y, Z))$ iff $x \trianglelefteq y \circ z$.

In addition, if $\mathbf{qVal}(X)$ and $\mathbf{qBound}(X, Y, Z)$ can be chosen as existential constraints of the form $\exists X_1 \dots \exists X_n (B_1 \wedge \dots \wedge B_m)$ —where B_j ($1 \leq j \leq m$) are atomic—we say that \mathcal{D} is *existentially expressible* in \mathcal{C} . \square

It can be proved that $\mathcal{B}, \mathcal{U}, \mathcal{W}$ and any qualification domain built from these with the help of \otimes are existentially expressible in any constraint domain \mathcal{C} that includes the basic values and computational features of \mathcal{R} .

Admissible triples $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ consist of a constraint domain \mathcal{C} , a qualification domain \mathcal{D} and a proximity relation $\mathcal{S} : S \times S \rightarrow D$ —where D is the carrier set of \mathcal{D} and S is the set of all variables, basic values and signature symbols available in \mathcal{C} —satisfying the following properties:

- $\forall x \in S : \mathcal{S}(x, x) = \mathbf{t}$ (reflexivity).
- $\forall x, y \in S : \mathcal{S}(x, y) = \mathcal{S}(y, x)$ (symmetry).
- Some additional technical conditions explained in (Rodríguez-Artalejo and Romero-Díaz 2010b).

A proximity relation \mathcal{S} is called *similarity* iff it satisfies the additional property $\forall x, y, z \in S : \mathcal{S}(x, z) \trianglerighteq \mathcal{S}(x, y) \sqcap \mathcal{S}(y, z)$ (transitivity). The scheme SQCLP has instances SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) where $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ is an admissible triple.

A SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program is a set \mathcal{P} of *qualified program rules* (also called *qualified clauses*) $C : A \stackrel{\alpha}{\leftarrow} B_1 \# w_1, \dots, B_m \# w_m$, where A is a defined atom, $\alpha \in D \setminus \{\mathbf{b}\}$ is called the *attenuation factor* of the clause and each $B_j \# w_j$ ($1 \leq j \leq m$) is an atom B_j annotated with a so-called *threshold value* $w_j \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$. The intended meaning of C is as follows: if for all $1 \leq j \leq m$ one has $B_j \# e_j$ (meaning that B_j holds with qualification value e_j) for some $e_j \trianglerighteq^? w_j$, then $A \# d$ (meaning that A holds with qualification value d) can be inferred for any $d \in D \setminus \{\mathbf{b}\}$ such that $d \trianglelefteq \alpha \circ \bigcap_{j=1}^m e_j$. By convention, $e_j \trianglerighteq^? w_j$ means $e_j \trianglerighteq w_j$ if $w_j \neq ?$ and is identically true otherwise. In practice threshold values equal to “?” and attenuation values equal to \mathbf{t} can be omitted.

Figure 1 shows a simple SQCLP($\mathcal{S}_s, \mathcal{U}, \mathcal{R}$)-program \mathcal{P}_s which illustrates the expressivity of the SQCLP scheme to deal with problems involving flexible information retrieval. Predicate *search* can be used to answer queries asking for books in the library matching some desired language, genre and reader level. Predicate *guessRdrLvl* takes advantage of attenuation factors to encode heuristic rules to

```

% Book representation: book( ID, Title, Author, Lang, Genre, VocLvl, Pages ).
1 library([ book(1, 'Tintin', 'Hergé', french, comic, easy, 65),
2          book(2, 'Dune', 'F.P. Herbert', english, sciFi, medium, 345),
3          book(3, 'Kritik der reinen Vernunft', 'I. Kant', german, philosophy, difficult, 1011),
4          book(4, 'Beim Hauten der Zwiebel', 'G. Grass', german, biography, medium, 432) ])

% Auxiliary predicate for computing list membership:
5 member(B, [B|_])
6 member(B, [_|T]) ← member(B, T)

% Predicates for getting the explicit attributes of a given book:
7 getId(book(ID, _Title, _Author, _Lang, _Genre, _VocLvl, _Pages), ID)
8 getTitle(book(_ID, Title, _Author, _Lang, _Genre, _VocLvl, _Pages), Title)
9 getAuthor(book(_ID, _Title, Author, _Lang, _Genre, _VocLvl, _Pages), Author)
10 getLanguage(book(_ID, _Title, _Author, Lang, _Genre, _VocLvl, _Pages), Lang)
11 getGenre(book(_ID, _Title, _Author, _Lang, Genre, _VocLvl, _Pages), Genre)
12 getVocLvl(book(_ID, _Title, _Author, _Lang, _Genre, VocLvl, _Pages), VocLvl)
13 getPages(book(_ID, _Title, _Author, _Lang, _Genre, _VocLvl, Pages), Pages)

% Function for guessing the reader level of a given book:
14 guessRdrLvl(B, basic) ← getVocLvl(B, easy), getPages(B, N), N < 50
15 guessRdrLvl(B, intermediate) ←0.8 getVocLvl(B, easy), getPages(B, N), N ≥ 50
16 guessRdrLvl(B, basic) ←0.9 getGenre(B, children)
17 guessRdrLvl(B, proficiency) ←0.9 getVocLvl(B, difficult), getPages(B, N), N ≥ 200
18 guessRdrLvl(B, upper) ←0.8 getVocLvl(B, difficult), getPages(B, N), N < 200
19 guessRdrLvl(B, intermediate) ←0.8 getVocLvl(B, medium)
20 guessRdrLvl(B, upper) ←0.7 getVocLvl(B, medium)

% Function for answering a particular kind of user queries:
21 search(Lang, Genre, Level, Id) ← library(L)#1.0, member(B, L)#1.0,
22     getLanguage(B, Lang), getGenre(B, Genre),
23     guessRdrLvl(B, Level), getId(B, Id)#1.0

% Proximity relation  $\mathcal{S}_s$ :
24  $\mathcal{S}_s(\text{sciFi}, \text{fantasy}) = \mathcal{S}_s(\text{fantasy}, \text{sciFi}) = 0.9$ 
25  $\mathcal{S}_s(\text{adventure}, \text{fantasy}) = \mathcal{S}_s(\text{fantasy}, \text{adventure}) = 0.7$ 
26  $\mathcal{S}_s(\text{essay}, \text{philosophy}) = \mathcal{S}_s(\text{philosophy}, \text{essay}) = 0.8$ 
27  $\mathcal{S}_s(\text{essay}, \text{biography}) = \mathcal{S}_s(\text{biography}, \text{essay}) = 0.7$ 

```

Fig. 1. SQCLP($\mathcal{S}_s, \mathcal{U}, \mathcal{R}$)-program \mathcal{P}_s (Library with books in different languages)

compute reader levels on the basis of vocabulary level and other book features. The other predicates compute book features in the natural way, and the proximity relation \mathcal{S}_s allows flexibility in any unification (i.e. solving of equality constraints) arising during the invocation of the program predicates.

The declarative semantics of a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} relies on *qualified constrained atoms* (briefly *qc-atoms*) of the form $A\#d \leftarrow \Pi$, intended to assert

that the validity of atom A with qualification degree $d \in D$ is entailed by the constraint set Π . A qc-atom is called *defined*, *primitive* or *equational* according to the syntactic form of A ; and it is called *observable* iff $d \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable.

Program interpretations are defined as sets of observable qc-atoms which obey a natural closure condition. The results proved in (Rodríguez-Artalejo and Romero-Díaz 2010a) show two equivalent ways to characterize declarative semantics, using a fix-point approach and a proof-theoretical approach, respectively. For the purposes of the present paper it suffices to consider the proof theoretical approach, that relies on a formal inference system called *Proximity-based Qualified Constrained Horn Logic*—in symbols, $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ —intended to infer observable qc-atoms from \mathcal{P} and consisting of the three inference rules displayed in Figure 2. Rule **SQEA** depends on a relation $\approx_{d, \Pi}$ between terms that is defined in the following way: $t \approx_{d, \Pi} s$ iff there exist two terms \hat{t} and \hat{s} such that $\Pi \models_{\mathcal{C}} t == \hat{t}$, $\Pi \models_{\mathcal{C}} s == \hat{s}$ and $\mathbf{b} \neq d \trianglelefteq \mathcal{S}(\hat{t}, \hat{s})$. This allows to deduce equations from Π in a flexible way, taking the proximity relation \mathcal{S} into account. The reader is referred to (Rodríguez-Artalejo and Romero-Díaz 2010b) for more motivating comments on $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ and some technical properties of the $\approx_{d, \Pi}$ relation.

$$\begin{array}{l}
 \textbf{SQDA} \quad \frac{((t'_i == t_i \theta) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \\
 \text{if } (p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}, \quad \theta \text{ subst.}, \quad \mathcal{S}(p', p) = d_0 \neq \mathbf{b}, \\
 e_j \triangleright^? w_j \quad (1 \leq j \leq m) \text{ and } d \trianglelefteq \prod_{i=0}^n d_i \sqcap \alpha \circ \prod_{j=1}^m e_j. \\
 \\
 \textbf{SQEA} \quad \frac{}{(t == s) \# d \Leftarrow \Pi} \quad \text{if } t \approx_{d, \Pi} s. \quad \textbf{SQPA} \quad \frac{}{\kappa \# d \Leftarrow \Pi} \quad \text{if } \Pi \models_{\mathcal{C}} \kappa.
 \end{array}$$

Fig. 2. Proximity-based Qualified Constrained Horn Logic

We will write $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \varphi$ to indicate that φ can be deduced from \mathcal{P} in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$, and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}}^k \varphi$ in the case that the deduction can be performed with exactly k **SQDA** inference steps. As usual in formal inference systems, $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proofs can be represented as *proof trees* whose nodes correspond to qc-atoms, each node being inferred from its children by means of some $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ inference step. The following theorem, proved in (Rodríguez-Artalejo and Romero-Díaz 2010b), characterizes least program models in the scheme SQCLP. This result allows to use $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -derivability as a logical criterion for proving the semantic correctness of program transformations, as we will do in Section 4.

Theorem 2.1 (Logical characterization of least program models in SQCHL)

For any SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} , its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \varphi\}. \quad \square$$

Let us now discuss goals and their solutions. Goals for a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-

program \mathcal{P} have the form

$$G : A_1 \# W_1, \dots, A_m \# W_m \parallel W_1 \triangleright^? \beta_1, \dots, W_m \triangleright^? \beta_m$$

abbreviated as $(A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1\dots m}$. The $A_i \# W_i$ are called *annotated atoms*. The pairwise different variables $W_i \in \mathcal{W}ar$ are called *qualification variables*; they are taken from a set $\mathcal{W}ar$ assumed to be disjoint from the set $\mathcal{V}ar$ of data variables used in terms. The conditions $W_i \triangleright^? \beta_i$ (with $\beta_i \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$) are called *threshold conditions* and their intended meaning (relying on the notations ‘?’ and ‘ $\triangleright^?$ ’) is as already explained when introducing program clauses above. In the sequel, $\text{war}(o)$ will denote the set of all qualification variables occurring in the syntactic object o . In particular, for a goal G as displayed above, $\text{war}(G)$ denotes the set $\{W_i \mid 1 \leq i \leq m\}$. In the case $m = 1$ the goal is called *atomic*. The following definition relies on $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -derivability to provide a natural declarative notion of goal solution:

Definition 2.2 (Goal Solutions)

Assume a given $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} and a goal G for \mathcal{P} with the syntax displayed above. Then:

1. A *solution* for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a \mathcal{C} -substitution, $W\mu \in D \setminus \{\mathbf{b}\}$ for all $W \in \text{dom}(\mu)$, Π is a satisfiable and finite set of atomic \mathcal{C} -constraints and the following two conditions hold for all $i = 1 \dots m$: $W_i\mu = d_i \triangleright^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$. The set of all solutions for G w.r.t. \mathcal{P} is noted $\text{Sol}_{\mathcal{P}}(G)$.
2. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \text{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i \eta$ is a ground atom for all $i = 1 \dots m$. The set of all ground solutions for G w.r.t. \mathcal{P} is noted $\text{GSol}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
3. A ground solution $\langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \mu, \Pi \rangle$ iff there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\text{var}(G)} \sigma\nu$ and $W_i \rho \leq W_i \mu$ for $i = 1 \dots m$. \square

A possible goal G_s for the library program displayed in Figure 1 is

$$G_s : \text{search}(\text{german}, \text{essay}, \text{intermediate}, ID) \# W \parallel W \geq 0.65$$

and one solution for G_s is $\langle \{ID \mapsto 4\}, \{W \mapsto 0.7\}, \emptyset \rangle$. In this simple case, the constraint set Π within the solution is empty. Other examples of goal solutions can be found in (Rodríguez-Artalejo and Romero-Díaz 2010b) and Sections 4 and 5 below.

In practice, users of SQCLP languages will rely on some available *goal solving system* for computing goal solutions. The following definition specifies two important abstract properties of goal solving systems which will be taken as a reference for the implementation presented in this paper.

Definition 2.3 (Correct Abstract Goal Solving Systems)

An *abstract goal solving system* for $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ is any device that takes a program \mathcal{P} and a goal G as input and yields various triples $\langle \sigma, \mu, \Pi \rangle$, called *computed answers*, as outputs. Such a goal solving system is called:

1. *Sound* iff every computed answer is a solution $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$.

2. *Weakly complete* iff every ground solution $\langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is subsumed by some computed answer.
3. *Correct* iff it is both sound and weakly complete. \square

Every goal solving system for a SQCLP instance should be sound and ideally also weakly complete. In principle, goal solving systems with these properties for extensions of the classical LP paradigm can be formalized as extensions of the well-known SLD-resolution method (Lloyd 1987; Apt 1990). A sound and complete extensions of SLD-resolution for the CLP scheme can be found e.g. in (Jaffar et al. 1998), and several extensions of SLD resolution for LP languages aiming at uncertain reasoning SQCLP scheme have been mentioned in Section 1.

Our aim in this paper is to present an implementation based on a semantically correct program transformation from SQCLP into CLP, rather than developing a sound and complete extension of SLD resolution. Nevertheless, both our implementation and SLD-based approaches for SLP languages in the line of (Sessa 2002) must share the ability to solve unification problems w.r.t. to a proximity relation $\mathcal{S} : S \times S \rightarrow [0, 1]$ over signature symbols, which is assumed to be transitive in (Sessa 2002) but not in our setting. The lack of transitivity makes a crucial difference w.r.t. the behavior of unification algorithms. In the rest of this section we briefly discuss the problem by means of a simple example.

(Sessa 2002) presents a flexible unification algorithm for solving unification problems represented as systems of the form $S \parallel \alpha$, where S is a set of equations between terms and α is a certainty degree. A solution of such a system is any substitution θ which verifies $\mathcal{S}(s\theta, t\theta) \geq \alpha$ for all equations $s == t$ belonging to S . This notion of solution is consistent with the declarative semantics of the SQCLP scheme (more specifically, with Definition 2.2), even in the case that \mathcal{S} is a non-transitive proximity relation. Following a traditional approach, Sessa presents the flexible unification algorithm as set of transformation rules which convert systems $S \parallel \alpha$ into solved form systems which represent unifiers. The transformations are similar to those presented in e.g. Section 4.6 of (Baader and Nipkow 1998) for the case of classical syntactic unification, extended with suitable computations to update α during the process, taking the given similarity relation \mathcal{S} into account. One of the transformations allows to transform a system of the form $X == t, S \parallel \alpha$ into $S\{X \mapsto t\} \parallel \alpha$ (provided that X is not identical to t and does not occur in t , the so-called occurs check). Unfortunately, this transformation can lose solutions in case that \mathcal{S} is not transitive. Consider for instance the following example:

Example 2.1

Assume constants a, b, c and a non-transitive proximity relation \mathcal{S} such that $\mathcal{S}(a, b) = \mathcal{S}(b, a) = 0.7$; $\mathcal{S}(a, c) = \mathcal{S}(c, a) = 0.8$; $\mathcal{S}(b, c) = \mathcal{S}(c, b) = 0$. Then, the substitution $\theta = \{X \mapsto a\}$ is obviously a solution of the unification problem $X == b, X == c \parallel 0.7$. Nevertheless, the unification algorithm presented in (Sessa 2002) and related papers fails without computing any solution:

$$X == b, X == c \parallel 0.7 \implies X == c \{X \mapsto b\} \parallel 0.7 \implies \text{fail}$$

The second transformation step leads to *fail* because $X == c \{X \mapsto b\} \parallel 0.7$ is

the same as $b == c \parallel 0.7$ and $\mathcal{S}(b, c) = 0 < 0.7$. Should \mathcal{S} satisfy transitivity, then $\mathcal{S}(b, c) = \mathcal{S}(c, b) \geq 0.7$, and Sessa's unification algorithm would compute the unifier $\sigma = \{X \mapsto b\}$ as follows:

$$X == b, X == c \parallel 0.7 \implies X == c \{X \mapsto b\} \parallel 0.7 \implies \{X \mapsto b\} \parallel 0.7$$

Note that σ is more general than θ in the sense that $\mathcal{S}(\theta, \sigma\theta) = \mathcal{S}(\theta, \sigma) \geq 0.7$. Therefore this example does not contradict the completeness of Sessa's unification algorithm for the case of (transitive) similarity relations. \square

Even in the case that \mathcal{S} is transitive, we have found examples showing that a goal solving system based on Sessa's unification algorithm can fail to compute some valid solutions for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs whose clauses use attenuation factors other than **t**. The unification algorithm underlying the implementations presented in Section 5—based on the program transformations from Section 4—avoids the problematic transformation step $X == t, S \parallel \alpha \implies S\{X \mapsto t\} \parallel \alpha$, that might cause incompleteness; instead, Prolog's backtracking is used to implement the effect of a non-deterministic choice between several transformation steps $X == c(\bar{t}_n), S \parallel \alpha \implies X_1 == t_1, \dots, X_n == t_n, S\mu \parallel \alpha$, where X_1, \dots, X_n are fresh variables and $\mu = \{X \mapsto c'(\bar{X}_n)\}$ for some possible choice of c' such that $\mathcal{S}(c, c') \geq \alpha$.

As an optimization, our prototype system allows the user to use a directive whose effect is that the system avoids the backtracking search just discussed and implements just the effect of the transformation $X == t, S \parallel \alpha \implies S\{X \mapsto t\} \parallel \alpha$. When including this directive, the user runs the risk of losing some valid solutions. We conjecture that no incompleteness occurs in the case of SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs based on a transitive \mathcal{S} and whose clauses do not use attenuation factors other than **t**; i.e. SLP programs enriched with constraint solving.

3 The Schemes QCLP & CLP as Specializations of SQCLP

As discussed in the concluding section of (Rodríguez-Artalejo and Romero-Díaz 2010a), several specializations of the SQCLP scheme can be obtained by partial instantiation of its parameters. In particular, QCLP and CLP can be defined as schemes with instances:

$$\begin{aligned} \text{QCLP}(\mathcal{D}, \mathcal{C}) &=_{\text{def}} \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C}) \\ \text{CLP}(\mathcal{C}) &=_{\text{def}} \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{B}, \mathcal{C}) = \text{QCLP}(\mathcal{B}, \mathcal{C}) \end{aligned}$$

where \mathcal{S}_{id} is the *identity* proximity relation and \mathcal{B} is the qualification domain including just the two classical boolean values. As explained in the introduction, QCLP and CLP are the targets of the two program transformations to be developed in Section 4. In this brief section we provide an explicit description of the syntax and semantics of these two schemes, derived from their behavior as specializations of SQCLP.

3.1 Presentation of the QCLP Scheme

As already explained, the instances of QCLP can be defined by the equation $\text{QCLP}(\mathcal{D}, \mathcal{C}) = \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C})$. Due to the admissibility of the parameter triple

$\langle \mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C} \rangle$, the qualification domain \mathcal{D} must be (existentially) expressible in the constraint domain \mathcal{C} . Technically, the QCLP scheme can be seen as a common extension of the classical CLP scheme for Constraint Logic Programming (Jaffar and Lassez 1987; Jaffar et al. 1998) and the QLP scheme for Qualified Logic Programming originally introduced in (Rodríguez-Artalejo and Romero-Díaz 2008). Intuitively, QCLP programming behaves like SQCLP programming, except that proximity information other than the identity is not available for proving equalities.

Program clauses and observable qc-atoms in QCLP are defined in the same way as in SQCLP. The library program \mathcal{P}_s in Figure 1 becomes a QCLP(\mathcal{U}, \mathcal{R})-program \mathcal{P}'_s just by replacing \mathcal{S}_{id} for \mathcal{S} . Of course, \mathcal{P}'_s does not support flexible unification as it was the case with \mathcal{P}_s .

As explained in Section 2, the proof system consisting of the three displayed in Figure 2 characterizes the declarative semantics of a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} . In the particular case $\mathcal{S} = \mathcal{S}_{\text{id}}$, the inference rules specialize to those displayed in Figure 3, yielding a formal proof system called *Qualified Constrained Horn Logic* – in symbols, QCHL(\mathcal{D}, \mathcal{C}) – which characterizes the declarative semantics of a given QCLP(\mathcal{D}, \mathcal{C})-program \mathcal{P} . Note that rule **SQEA** depends on a relation \approx_Π between terms that is defined to behave the same as the specialization of $\approx_{d,\Pi}$ to the case $\mathcal{S} = \mathcal{S}_{\text{id}}$. It is easily checked that $t \approx_\Pi s$ does not depend on d and holds iff $\Pi \models_{\mathcal{C}} t == s$. Both $\approx_{d,\Pi}$ and \approx_Π allow to use the constraints within Π when deducing equations. However, $c(\bar{t}_n) \approx_\Pi c'(\bar{s}_n)$ never holds in the case that c and c' are not syntactically identical.

$$\begin{array}{l}
 \textbf{QDA} \quad \frac{((t'_i == t_i \theta) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{p(\bar{t}'_n) \# d \Leftarrow \Pi} \\
 \\
 \text{if } (p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}, \theta \text{ subst.}, \\
 e_j \triangleright^? w_j \ (1 \leq j \leq m) \text{ and } d \leq \prod_{i=1}^n d_i \sqcap \alpha \circ \prod_{j=1}^m e_j. \\
 \\
 \textbf{QEA} \quad \frac{}{(t == s) \# d \Leftarrow \Pi} \quad \text{if } t \approx_\Pi s. \qquad \textbf{QPA} \quad \frac{}{\kappa \# d \Leftarrow \Pi} \quad \text{if } \Pi \models_{\mathcal{C}} \kappa.
 \end{array}$$

Fig. 3. Qualified Constrained Horn Logic

SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) proof trees and the notations related to them can be naturally specialized to QCHL(\mathcal{D}, \mathcal{C}). In particular, we will use the notation $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \varphi$ (resp. $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}}^k \varphi$) to indicate that the qc-atom φ can be inferred in QCHL(\mathcal{D}, \mathcal{C}) from the program \mathcal{P} (resp. it can be inferred by using exactly k **QDA** inference steps). Theorem 2.1 also specializes to QCHL, yielding the following result:

Theorem 3.1 (Logical characterization of least program models in QCHL)

For any QCLP(\mathcal{D}, \mathcal{C})-program \mathcal{P} , its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \varphi\}. \quad \square$$

Concerning goals and their solutions, their specialization to the particular case $\mathcal{S} = \mathcal{S}_{\text{id}}$ leaves the syntax of goals G unaffected and leads to the following definition, almost identical to Definition 2.2:

Definition 3.1 (Goal Solutions in QCLP)

Assume a given QCLP(\mathcal{S}, \mathcal{D}) \mathcal{C} -program \mathcal{P} and a goal $G : (A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$. Then:

1. A *solution* for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a \mathcal{C} -substitution, $W\mu \in D \setminus \{\mathbf{b}\}$ for all $W \in \text{dom}(\mu)$, Π is a satisfiable and finite set of atomic \mathcal{C} -constraints, and the following two conditions hold for all $i = 1 \dots m$: $W_i\mu = d_i \triangleright^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A_i \sigma \# W_i \mu \Leftarrow \Pi$. The set of all solutions for G is noted $\text{Sol}_{\mathcal{P}}(G)$.
2. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \text{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i \eta$ is a ground atom for all $i = 1 \dots m$. The set of all ground solutions for G is noted $\text{GSol}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
3. A ground solution $\langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \mu, \Pi \rangle$ iff there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\text{var}(G)} \sigma \nu$ and $W_i \rho \trianglelefteq W_i \mu$ for $i = 1 \dots m$. \square

Finally, the notion of correct abstract goal solving system for SQCLP given in Definition 2.3 specializes to QCLP without any formal change. Therefore, we state no new definition at this point.

3.2 Presentation of the CLP Scheme

As already explained, the instances of CLP can be defined by the equation $\text{CLP}(\mathcal{C}) = \text{SQCLP}(\mathcal{S}_{\text{id}}, \mathcal{B}, \mathcal{C})$, or equivalently, $\text{CLP}(\mathcal{C}) = \text{QCLP}(\mathcal{B}, \mathcal{C})$. Due to the fixed choice $\mathcal{D} = \mathcal{B}$, the only qualification value $d \in D \setminus \{\mathbf{b}\}$ available for use as attenuation factor or threshold value is $d = \mathbf{t}$. Therefore, CLP can only include threshold values equal to ‘?’ and attenuation values equal to the top element $\mathbf{t} = \text{true}$ of \mathcal{B} . As explained in Section 2, such trivial threshold and attenuation values can be omitted, and CLP clauses can be written with the simplified syntax $A \leftarrow B_1, \dots, B_m$.

Since $\mathbf{t} = \text{true}$ is the only non-trivial qualification value available in CLP, qc-atoms $A \# d \Leftarrow \Pi$ are always of the form $A \# \text{true} \Leftarrow \Pi$ and can be written as $A \Leftarrow \Pi$. Moreover, all the side conditions for the inference rule **QDA** in Figure 3 become trivial when specialized to the case $\mathcal{D} = \mathcal{B}$. Therefore, the specialization of QCHL(\mathcal{D}, \mathcal{C}) to the case $\mathcal{D} = \mathcal{B}$ leads to the formal proof system called *Constrained Horn Logic* – in symbols, $\text{CHL}(\mathcal{C})$ – consisting of the three inference rules displayed in Figure 4, which characterizes the declarative semantics of a given CLP(\mathcal{C})-program \mathcal{P} .

QCHL(\mathcal{D}, \mathcal{C}) proof trees and the notations related to them can be naturally specialized to CHL(\mathcal{C}). In particular, we will use the notation $\mathcal{P} \vdash_{\varphi}$ (resp. $\mathcal{P} \vdash_k \varphi$) to indicate that the qc-atom φ can be inferred in CHL(\mathcal{C}) from the program \mathcal{P} (resp. it can be inferred by using exactly k **DA** inference steps). Theorem 3.1 also specializes to CHL, yielding the following result:

$$\begin{array}{l}
\mathbf{DA} \quad \frac{((t'_i == t_i \theta) \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \Leftarrow \Pi)_{j=1 \dots m}}{p(\bar{t}'_n) \Leftarrow \Pi} \\
\text{if } (p(\bar{t}'_n) \Leftarrow B_1, \dots, B_m) \in \mathcal{P} \text{ and } \theta \text{ subst.} \\
\mathbf{EA} \quad \frac{}{(t == s) \Leftarrow \Pi} \quad \text{if } t \approx_{\Pi} s. \qquad \mathbf{PA} \quad \frac{}{\kappa \Leftarrow \Pi} \quad \text{if } \Pi \models_{\mathcal{C}} \kappa.
\end{array}$$

Fig. 4. Constrained Horn Logic

Theorem 3.2 (Logical characterization of least program models in CHL)

For any CLP(\mathcal{C})-program \mathcal{P} , its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{ \varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\varphi} \}. \quad \square$$

Concerning goals and their solutions, their specialization to the scheme CLP leads to the following definition:

Definition 3.2 (Goals and their Solutions in CLP)

Assume a given CLP(\mathcal{C})-program \mathcal{P} . Then:

1. Goals for \mathcal{P} have the form $G : A_1, \dots, A_m$, abbreviated as $(A_i)_{i=1 \dots m}$, where A_i ($1 \leq i \leq m$) are atoms.
2. A *solution* for a goal G is any pair $\langle \sigma, \Pi \rangle$ such that σ is a \mathcal{C} -substitution, Π is a satisfiable and finite set of atomic \mathcal{C} -constraints, and $\mathcal{P} \vdash_{\mathcal{C}} A_i \sigma \Leftarrow \Pi$ holds for all $i = 1 \dots m$. The set of all solutions for G is noted $\text{Sol}_{\mathcal{P}}(G)$.
3. A solution $\langle \eta, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \text{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i \eta$ is a ground atom for all $i = 1 \dots m$. The set of all ground solutions for G is noted $\text{GSol}_{\mathcal{P}}(G)$. Obviously, $\text{GSol}_{\mathcal{P}}(G) \subseteq \text{Sol}_{\mathcal{P}}(G)$.
4. A ground solution $\langle \eta, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \Pi \rangle$ iff there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\text{var}(G)} \sigma \nu$. \square

The notion of correct abstract goal solving system for SQCFLLP given in Definition 2.3 specializes to CLP with only minor formal changes, as follows:

Definition 3.3 (Correct Abstract Goal Solving Systems for CLP)

A *goal solving system* for CLP(\mathcal{C}) is any effective procedure which takes a program \mathcal{P} and a goal G as input and yields various pairs $\langle \sigma, \Pi \rangle$, called *computed answers*, as outputs. Such a goal solving system is called:

1. *Sound* iff every computed answer is a solution $\langle \sigma, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$.
2. *Weakly complete* iff every ground solution $\langle \eta, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ is subsumed by some computed answer.
3. *Correct* iff it is both sound and weakly complete. \square

We close this Subsection with a technical lemma that will be useful for proving some results in Subsection 4.2:

Lemma 3.1

Assume an existential \mathcal{C} -constraint $\pi(\overline{X}_n) = \exists Y_1 \dots \exists Y_k (B_1 \wedge \dots \wedge B_m)$ with free variables \overline{X}_n and a given $\text{CLP}(\mathcal{C})$ -program \mathcal{P} including the clause $C : p(\overline{X}_n) \leftarrow B_1, \dots, B_m$, where $p \in DP^n$ does not occur at the head of any other clause of \mathcal{P} . Then, for any n -tuple \bar{t}_n of \mathcal{C} -terms and any finite and satisfiable $\Pi \subseteq \text{Con}_{\mathcal{C}}$, one has:

1. $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi) \implies \Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$, where $\pi(\bar{t}_n)$ stands for the result of applying the substitution $\{\overline{X}_n \mapsto \bar{t}_n\}$ to $\pi(\overline{X}_n)$.
2. The opposite implication $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \implies \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ holds if \bar{t}_n is a ground term tuple. Note that for ground \bar{t}_n the constraint entailment $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$ simply means that $\pi(\bar{t}_n)$ is true in \mathcal{C} .
3. $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \implies \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ may fail if \bar{t}_n is not a ground term tuple.

Proof

We prove each item separately:

1. Assume $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$. Note that C is the only clause for p in \mathcal{P} and that each atom B_j in C 's body is an atomic constraint. Therefore, the $\text{CHL}(\mathcal{C})$ proof must use a **DA** step based on an instance $C\theta$ of clause C such that $\Pi \models_{\mathcal{C}} t_i = X_i\theta$ holds for all $1 \leq i \leq n$ and $\Pi \models_{\mathcal{C}} B_j\theta$ holds for all $1 \leq j \leq m$. These conditions and the syntactic form of $\pi(\overline{X}_n)$ obviously imply $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$.
2. Assume now $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$ and \bar{t}_n ground. Then $\pi(\bar{t}_n)$ is true in \mathcal{C} , and due to the syntactic form of $\pi(\overline{X}_n)$, there must be some substitution θ such that $X_i\theta = t_i$ (syntactic identity) for all $1 \leq i \leq n$ and $B_j\theta$ is ground and true in \mathcal{C} for all $1 \leq j \leq m$. Trivially, $\Pi \models_{\mathcal{C}} t_i = X_i\theta$ holds for all $1 \leq i \leq n$ and $\Pi \models_{\mathcal{C}} B_j\theta$ also holds for all $1 \leq j \leq m$. Then, it is obvious that $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ can be proved by using a **DA** step based on the instance $C\theta$ of clause C .
3. We prove that $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \implies \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ can fail if \bar{t}_n is not ground by presenting a counterexample based on the constraint domain \mathcal{R} , using the syntax for \mathcal{R} -constraints explained in (Rodríguez-Artalejo and Romero-Díaz 2010b). Consider the existential \mathcal{R} -constraint $\pi(X) = \exists Y (op_+(Y, Y, X))$, and a $\text{CLP}(\mathcal{R})$ -program \mathcal{P} including the clause $C : p(X) \leftarrow op_+(Y, Y, X)$ and no other occurrence of the defined predicate symbol p . Consider also $\Pi = \{cp_{\geq}(X, 0.0)\}$ and $t = X$. Then $\Pi \models_{\mathcal{R}} \pi(X)$ is obviously true, because any real number $x \geq 0.0$ satisfies $\exists Y (op_+(Y, Y, x))$ in \mathcal{R} . However, there is no \mathcal{R} -term s such that $\Pi \models_{\mathcal{R}} op_+(s, s, X)$, and therefore there is no instance $C\theta$ of clause C that can be used to prove $\mathcal{P} \vdash_{\mathcal{C}} (p(X) \Leftarrow \Pi)$ by applying a **DA** step. \square

4 Implementation by Program Transformation

The purpose of this section is to introduce a program transformation that transforms $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ programs and goals into semantically equivalent $\text{CLP}(\mathcal{C})$ programs and goals. This transformation is performed as the composition of the two following specific transformations:

1. $\text{elim}_{\mathcal{S}}$ — Eliminates the proximity relation \mathcal{S} of arbitrary $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ programs and goals, producing equivalent $\text{QCLP}(\mathcal{D}, \mathcal{C})$ programs and goals.
2. $\text{elim}_{\mathcal{D}}$ — Eliminates the qualification domain \mathcal{D} of arbitrary $\text{QCLP}(\mathcal{D}, \mathcal{C})$ programs and goals, producing equivalent $\text{CLP}(\mathcal{C})$ programs and goals.

Thus, given a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} —resp. $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -goal G —, the composition of the two transformations will produce an equivalent $\text{CLP}(\mathcal{C})$ -program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$ —resp. $\text{CLP}(\mathcal{C})$ -goal $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G))$ —.

Example 4.1 (Running example: $\text{SQCLP}(\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R})$ -program \mathcal{P}_r)

As a running example for this section, consider the $\text{SQCLP}(\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R})$ -program \mathcal{P}_r as follows:

$$\begin{array}{ll}
r_1 & \text{famous}(\text{sha}) \xleftarrow{(0.9,1)} \\
r_2 & \text{wrote}(\text{sha}, \text{kle}) \xleftarrow{(1,1)} \\
r_3 & \text{wrote}(\text{sha}, \text{hamlet}) \xleftarrow{(1,1)} \\
r_4 & \text{good_work}(G) \xleftarrow{(0.75,3)} \text{famous}(A) \# (0.5,100), \text{authored}(A, G) \\
s_1 & \mathcal{S}_r(\text{wrote}, \text{authored}) = \mathcal{S}_r(\text{authored}, \text{wrote}) = (0.9,0) \\
s_2 & \mathcal{S}_r(\text{kle}, \text{kli}) = \mathcal{S}_r(\text{kli}, \text{kle}) = (0.8,2)
\end{array}$$

where the constants *shakespeare*, *king_lear* and *king_liar* have been respectively replaced, for clarity purposes in the subsequent examples, by *sha*, *kle* and *kli*.

In addition, consider the $\text{SQCLP}(\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R})$ -goal G_r as follows:

$$\text{good_work}(X) \# W \parallel W \triangleright^? (0.5,10)$$

We will illustrate the two transformation by showing, in subsequent examples, the program clauses of $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ and $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ and the goals $\text{elim}_{\mathcal{S}}(G_r)$ and $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G_r))$. \square

The next two subsections explain each transformation in detail.

4.1 Transforming SQCLP into QCLP

In this subsection we assume that the triple $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ is admissible. In the sequel we say that a defined predicate symbol $p \in DP^n$ is *affected* by a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} iff $\mathcal{S}(p, p') \neq \mathbf{b}$ for some p' occurring in \mathcal{P} . We also say that an atom A is *relevant* for \mathcal{P} iff some of the three following cases hold: a) A is an equation $t = s$; b) A is a primitive atom κ ; or c) A is a defined atom $p(\bar{t}_n)$ such that p is affected by \mathcal{P} .

As a first step towards the definition of the first program transformation $\text{elim}_{\mathcal{S}}$, we define a set $EQ_{\mathcal{S}}$ of $\text{QCLP}(\mathcal{D}, \mathcal{C})$ program clauses that emulates the behavior of equations in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$. The following definition assumes that the binary predicate symbol $\sim \in DP^2$ (used in infix notation) and the nullary predicate symbols $\text{pay}_{\lambda} \in DP^0$ are not affected by \mathcal{P} .

Definition 4.1

We define EQ_S as the following QCLP(\mathcal{D}, \mathcal{C})-program:

$$\begin{aligned}
 EQ_S =_{\text{def}} \{ & X \sim Y \stackrel{t}{\leftarrow} (X == Y) \#? \} \\
 & \cup \{ u \sim u' \stackrel{t}{\leftarrow} \text{pay}_\lambda \#? \mid u, u' \in B_{\mathcal{C}} \text{ and } \mathcal{S}(u, u') = \lambda \neq \mathbf{b} \} \\
 & \cup \{ c(\overline{X}_n) \sim c'(\overline{Y}_n) \stackrel{t}{\leftarrow} \text{pay}_\lambda \#?, ((X_i \sim Y_i) \#?)_{i=1 \dots n} \mid c, c' \in DC^n \\
 & \text{and } \mathcal{S}(c, c') = \lambda \neq \mathbf{b} \} \\
 & \cup \{ \text{pay}_\lambda \stackrel{\lambda}{\leftarrow} \mid \text{for each } \lambda \in D \setminus \{\mathbf{b}\} \}. \quad \square
 \end{aligned}$$

The following lemma shows the relation between the semantics of equations in SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$) and the behavior of the binary predicate symbol ' \sim ' defined by EQ_S in QCHL(\mathcal{D}, \mathcal{C}).

Lemma 4.1

Consider any two arbitrary terms t and s ; EQ_S defined as in Definition 4.1; and a satisfiable finite set Π of \mathcal{C} -constraints. Then, for every $d \in D \setminus \{\mathbf{b}\}$:

$$t \approx_{d, \Pi} s \iff EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \#d \Leftarrow \Pi .$$

Proof

We separately prove each implication.

[\implies] Assume $t \approx_{d, \Pi} s$. Then, there are two terms \hat{t}, \hat{s} such that:

$$(1) t \approx_{\Pi} \hat{t} \quad (2) s \approx_{\Pi} \hat{s} \quad (3) \hat{t} \approx_d \hat{s}$$

We use structural induction on the form of the term \hat{t} .

- $\hat{t} = Z, Z \in \text{Var}$. From (3) we have $\hat{s} = Z$. Then (1) and (2) become $t \approx_{\Pi} Z$ and $s \approx_{\Pi} Z$, therefore $t \approx_{\Pi} s$. Now $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \#d \Leftarrow \Pi$ can be proved with a proof tree rooted by a **QDA** step of the form:

$$\frac{(t == X\theta) \#t \Leftarrow \Pi \quad (s == Y\theta) \#t \Leftarrow \Pi \quad (X == Y) \theta \#t \Leftarrow \Pi}{(t \sim s) \#d \Leftarrow \Pi}$$

using the clause $X \sim Y \stackrel{t}{\leftarrow} (X == Y) \#? \in EQ_S$ instantiated by the substitution $\theta = \{X \mapsto t, Y \mapsto s\}$. Therefore the three premises can be derived from EQ_S with **QEA** steps since $t \approx_{\Pi} t, s \approx_{\Pi} s$ and $t \approx_{\Pi} s$, respectively. Checking the side conditions of all inference steps is straightforward.

- $\hat{t} = u, u \in B_{\mathcal{C}}$. From (3) we have $\hat{s} = u'$ for some $u' \in B_{\mathcal{C}}$ such that $d \leq \lambda = \mathcal{S}(u, u')$. Then (1) and (2) become $t \approx_{\Pi} u$ and $s \approx_{\Pi} u'$, which allow to build a proof of $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \#d \Leftarrow \Pi$ by means of a **QDA** step using the clause $u \sim u' \stackrel{t}{\leftarrow} \text{pay}_\lambda \#?$.
- $\hat{t} = c, c \in DC^0$. From (3) we have $\hat{s} = c'$ for some $c' \in DC^0$ such that $d \leq \lambda = \mathcal{S}(c, c')$. Then (1) and (2) become $t \approx_{\Pi} c$ and $s \approx_{\Pi} c'$, which allow us to build a proof of $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \#d \Leftarrow \Pi$ by means of a **QDA** step using the clause $c \sim c' \stackrel{t}{\leftarrow} \text{pay}_\lambda \#?$.
- $\hat{t} = c(\overline{t}_n), c \in DC^n$ with $n > 0$. In this case, and because of (3), we can assume $\hat{s} = c'(\overline{s}_n)$ for some $c' \in DC^n$ satisfying $d \leq d_0 =_{\text{def}} \mathcal{S}(c, c')$ and $d \leq d_i =_{\text{def}} \mathcal{S}(t_i, s_i)$

for $i = 1 \dots n$. Then $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ with a proof tree rooted by a **QDA** step of the form:

$$\frac{\begin{array}{ll} (t == c(\bar{t}_n)) \# \mathbf{t} \Leftarrow \Pi & \text{pay}_{d_0} \# d_0 \Leftarrow \Pi \\ (s == c'(\bar{s}_n)) \# \mathbf{t} \Leftarrow \Pi & ((t_i \sim s_i) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \end{array}}{(t \sim s) \# d \Leftarrow \Pi}$$

using the EQ_S clause $C : c(\bar{X}_n) \sim c'(\bar{Y}_n) \stackrel{\mathbf{t}}{\Leftarrow} \text{pay}_{d_0} \# ?, ((X_i \sim Y_i) \# ?)_{i=1 \dots n}$ instantiated by the substitution $\theta = \{X_1 \mapsto t_1, Y_1 \mapsto s_1, \dots, X_n \mapsto t_n, Y_n \mapsto s_n\}$. Note that C has attenuation factor \mathbf{t} and threshold values $?$ at the body. Therefore, the side conditions of the **QDA** step boil down to $d \trianglelefteq d_i$ ($1 \leq i \leq n$) which are true by assumption. It remains to prove that each premise of the **QDA** step can be derived from EQ_S in $\text{QCHL}(\mathcal{D}, \mathcal{C})$:

- $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t == c(\bar{t}_n)) \# \mathbf{t} \Leftarrow \Pi$ and $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (s == c'(\bar{s}_n)) \# \mathbf{t} \Leftarrow \Pi$ are trivial consequences of $t \approx_{\Pi} c(\bar{t}_n)$ and $s \approx_{\Pi} c'(\bar{s}_n)$, respectively. In both cases, the $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proofs consist of one single **QEA** step.
- $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} \text{pay}_{d_0} \# d_0 \Leftarrow \Pi$ can be proved using the clause $\text{pay}_{d_0} \stackrel{d_0}{\Leftarrow} \in EQ_S$ in one single **QDA** step.
- $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t_i \sim s_i) \# d_i \Leftarrow \Pi$ for $i = 1 \dots n$. For each i , we observe that $t_i \approx_{d_i, \Pi} s_i$ holds because of $\hat{t}_i = t_i$, $\hat{s}_i = s_i$ which satisfy $t_i \approx_{\Pi} \hat{t}_i$, $s_i \approx_{\Pi} \hat{s}_i$ and $\hat{t}_i \approx_{d_i} \hat{s}_i$. Since $\hat{t}_i = t_i$ is a subterm of $\hat{t} = c(\bar{t}_n)$, the inductive hypothesis can be applied.

[\Leftarrow] Let T be a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ -proof tree witnessing $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$. We prove $t \approx_{d, \Pi} s$ reasoning by induction on the number $n = \|T\|$ of nodes in T that represent conclusions of **QDA** inference steps. Note that all the program clauses belonging to EQ_S define either the binary predicate symbol ' \sim ' or the nullary predicates pay_{λ} .

Basis ($n = 1$).

In this case we have for the **QDA** inference step that there can be used three possible EQ_S clauses:

1. The program clause is $X \sim Y \stackrel{\mathbf{t}}{\Leftarrow} (X == Y) \# ?$. Then the **QDA** inference step must be of the form:

$$\frac{(t == t') \# d_1 \Leftarrow \Pi \quad (s == s') \# d_2 \Leftarrow \Pi \quad (t' == s') \# e_1 \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap e_1$. The proof of the three premises must use the **QEA** inference rule. Because of the conditions of this inference rule we have $t \approx_{\Pi} t'$, $s \approx_{\Pi} s'$ and $t' \approx_{\Pi} s'$. Therefore $t \approx_{\Pi} s$ is clear. Then $t \approx_{d, \Pi} s$ holds by taking $\hat{t} = \hat{s} = t$ because, trivially, $t \approx_{\Pi} \hat{t}$, $s \approx_{\Pi} \hat{s}$ and $\hat{t} \approx_d \hat{s}$.

2. The program clause is $u \sim u' \stackrel{\mathbf{t}}{\Leftarrow} \text{pay}_{\lambda} \# ?$ with $u, u' \in B_C$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$. The **QDA** inference step must be of the form:

$$\frac{(t == u) \# d_1 \Leftarrow \Pi \quad (s == u') \# d_2 \Leftarrow \Pi \quad \text{pay}_{\lambda} \# e_1 \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap e_1$. Due to the forms of the **QEA** inference rule and the EQ_S clause $\text{pay}_\lambda \xleftarrow{\lambda}$, we can assume without loss of generality that $d_1 = d_2 = \mathbf{t}$ and $e_1 = \lambda$. Therefore $d \trianglelefteq \lambda$. Moreover, the $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proofs of the first two premises must use **QEA** inferences. Consequently we have $t \approx_\Pi u$ and $s \approx_\Pi u'$. These facts and $u \approx_d u'$ imply $t \approx_{d, \Pi} s$.

3. The program clause is $c \sim c' \xleftarrow{\mathbf{t}} \text{pay}_\lambda \#?$ with $c, c' \in DC^0$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{b}$. The **QDA** inference step must be of the form:

$$\frac{(t == c) \# d_1 \Leftarrow \Pi \quad (s == c') \# d_2 \Leftarrow \Pi \quad \text{pay}_\lambda \# e_1 \Leftarrow \Pi}{(t \sim s) \# d \Leftarrow \Pi}$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap e_1$. Due to the forms of the **QEA** inference rule and the EQ_S clause $\text{pay}_\lambda \xleftarrow{\lambda}$, we can assume without loss of generality that $d_1 = d_2 = \mathbf{t}$ and $e_1 = \lambda$. Therefore $d \trianglelefteq \lambda$. Moreover, the $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proofs of the first two premises must use **QEA** inferences. Consequently we have $t \approx_\Pi c$ and $s \approx_\Pi c'$. These facts and $c \approx_d c'$ imply $t \approx_{d, \Pi} s$.

Inductive step ($n > 1$).

In this case t and s must be of the form $t = c(\bar{t}_n)$ and $s = c'(\bar{s}_n)$. The EQ_S clause used in the **QDA** inference step at the root must be of the form:

$$c(\bar{X}_n) \sim c'(\bar{Y}_n) \xleftarrow{\mathbf{t}} \text{pay}_{d_0} \#?, ((X_i \sim Y_i) \#?)_{i=1 \dots n}$$

with $\mathcal{S}(c, c') = d_0 \neq \mathbf{b}$. The inference step at the root will be:

$$\frac{(t == c(\bar{t}_n)) \# d_1 \Leftarrow \Pi \quad \text{pay}_{d_0} \# e_0 \Leftarrow \Pi}{(s == c'(\bar{s}_n)) \# d_2 \Leftarrow \Pi \quad ((t_i \sim s_i) \# e_i \Leftarrow \Pi)_{i=1 \dots n}} \quad (t \sim s) \# d \Leftarrow \Pi$$

with $d \trianglelefteq d_1 \sqcap d_2 \sqcap \prod_{i=1}^n e_i$. Due to the forms of the EQ_S clause $\text{pay}_{d_0} \xleftarrow{d_0}$ and the **QEA** inference rule there is no loss of generality in assuming $d_1 = d_2 = \mathbf{t}$ and $e_0 = d_0$, therefore we have $d \trianglelefteq d_0 \sqcap \prod_{i=1}^n e_i$. By the inductive hypothesis $t_i \approx_{e_i, \Pi} s_i$ ($1 \leq i \leq n$), i.e. there are constructor terms \hat{t}_i, \hat{s}_i such that $t_i \approx_\Pi \hat{t}_i$, $s_i \approx_\Pi \hat{s}_i$ and $\hat{t}_i \approx_{e_i} \hat{s}_i$ for $i = 1 \dots n$. Thus, we can build $\hat{t} = c(\hat{t}_1, \dots, \hat{t}_n)$ and $\hat{s} = c'(\hat{s}_1, \dots, \hat{s}_n)$ having $t \approx_{d, \Pi} s$ because:

- $t \approx_\Pi \hat{t}$, i.e. $c(\bar{t}_n) \approx_\Pi c(\bar{\hat{t}}_n)$, by decomposition since $t_i \approx_\Pi \hat{t}_i$.
- $s \approx_\Pi \hat{s}$, i.e. $c'(\bar{s}_n) \approx_\Pi c'(\bar{\hat{s}}_n)$, again by decomposition since $s_i \approx_\Pi \hat{s}_i$.
- $\hat{t} \approx_d \hat{s}$, since $d \trianglelefteq d_0 \sqcap \prod_{i=1}^n e_i \trianglelefteq \mathcal{S}(c, c') \sqcap \prod_{i=1}^n \mathcal{S}(\hat{t}_i, \hat{s}_i) = \mathcal{S}(\hat{t}, \hat{s})$. \square

We are now ready to define elim_S acting over programs and goals.

Definition 4.2

Assume a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} and a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -goal G for \mathcal{P} whose atoms are all relevant for \mathcal{P} . Then we define:

1. For each atom A , let A_\sim be $t \sim s$ if $A : t == s$; otherwise let A_\sim be A .
2. For each clause $C : (p(\bar{t}_n) \xleftarrow{\alpha} \bar{B}) \in \mathcal{P}$ let $\hat{\mathcal{C}}_S$ be the set of $\text{QCLP}(\mathcal{D}, \mathcal{C})$ clauses consisting of:

- The clause $\hat{C} : (\hat{p}_C(\bar{t}_n) \leftarrow^\alpha \bar{B}_\sim)$, where $\hat{p}_C \in DP^n$ is not affected by \mathcal{P} (chosen in a different way for each C) and \bar{B}_\sim is obtained from \bar{B} by replacing each atom A occurring in \bar{B} by A_\sim .
 - A clause $p'(\bar{X}_n) \leftarrow^t \text{pay}_\lambda \#?$, $((X_i \sim t_i) \#?)_{i=1\dots n}$, $\hat{p}_C(\bar{t}_n) \#?$ for each $p' \in DP^n$ such that $\mathcal{S}(p, p') = \lambda \neq \mathbf{b}$. Here, \bar{X}_n must be chosen as n pairwise different variables not occurring in the clause C .
3. $\text{elim}_\mathcal{S}(\mathcal{P})$ is the QCLP(\mathcal{D}, \mathcal{C})-program $EQ_\mathcal{S} \cup \hat{\mathcal{P}}_\mathcal{S}$ where $\hat{\mathcal{P}}_\mathcal{S} =_{\text{def}} \bigcup_{C \in \mathcal{P}} \hat{C}_\mathcal{S}$.
 4. $\text{elim}_\mathcal{S}(G)$ is the QCLP(\mathcal{D}, \mathcal{C})-goal G_\sim obtained from G by replacing each atom A occurring in G by A_\sim . \square

The following example illustrates the transformation $\text{elim}_\mathcal{S}$.

Example 4.2 (Running example: QCLP($\mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-program $\text{elim}_\mathcal{S}(\mathcal{P}_r)$)

Consider the SQCLP($\mathcal{S}_r, \mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-program \mathcal{P}_r and the goal G_r for \mathcal{P}_r as presented in Example 4.1. The transformed QCLP($\mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-program $\text{elim}_\mathcal{S}(\mathcal{P}_r)$ is as follows:

$$\begin{array}{ll}
\hat{r}_1 & \hat{famous}_{R_1}(sha) \leftarrow^{\langle 0.9, 1 \rangle} \\
R_{1.1} & famous(X) \leftarrow \text{pay}_t, X \sim sha, \hat{famous}_{R_1}(sha) \\
\hat{r}_2 & \hat{wrote}_{R_2}(sha, kle) \leftarrow^{\langle 1, 1 \rangle} \\
R_{2.1} & wrote(X, Y) \leftarrow \text{pay}_t, X \sim sha, Y \sim kle, \hat{wrote}_{R_2}(sha, kle) \\
R_{2.2} & authored(X, Y) \leftarrow \text{pay}_{\langle 0.9, 0 \rangle}, X \sim sha, Y \sim kle, \hat{wrote}_{R_2}(sha, kle) \\
\hat{r}_3 & \hat{wrote}_{R_3}(sha, hamlet) \leftarrow^{\langle 1, 1 \rangle} \\
R_{3.1} & wrote(X, Y) \leftarrow \text{pay}_t, X \sim sha, Y \sim hamlet, \hat{wrote}_{R_3}(sha, hamlet) \\
R_{3.2} & authored(X, Y) \leftarrow \text{pay}_{\langle 0.9, 0 \rangle}, X \sim sha, Y \sim hamlet, \hat{wrote}_{R_3}(sha, hamlet) \\
\hat{r}_4 & \hat{good_work}_{R_4}(G) \leftarrow^{\langle 0.75, 3 \rangle} famous(A) \#(0.5, 100), authored(A, G) \\
R_{4.1} & good_work(X) \leftarrow \text{pay}_t, X \sim G, \hat{good_work}_{R_4}(G)
\end{array}$$

<p>% Program clauses for \sim:</p> <p>$X \sim Y \leftarrow X == Y$</p> <p>$kle \sim kli \leftarrow \text{pay}_{\langle 0.8, 2 \rangle}$</p> <p>[...]</p>	<p>% Program clauses for pay:</p> <p>$\text{pay}_t \leftarrow$</p> <p>$\text{pay}_{\langle 0.9, 0 \rangle} \leftarrow^{\langle 0.9, 0 \rangle}$</p> <p>$\text{pay}_{\langle 0.8, 2 \rangle} \leftarrow^{\langle 0.8, 2 \rangle}$</p>
---	---

Finally, the goal $\text{elim}_\mathcal{S}(G_r)$ for $\text{elim}_\mathcal{S}(\mathcal{P}_r)$ is as follows:

$$good_work(X) \# W \parallel W \triangleright^? (0.5, 10) \quad \square$$

The next theorem proves the semantic correctness of the program transformation.

Theorem 4.1

Consider a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} , an atom A relevant for \mathcal{P} , a qualification value $d \in D \setminus \{\mathbf{b}\}$ and a satisfiable finite set of \mathcal{C} -constraints Π . Then, the following two statements are equivalent:

1. $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$
2. $\text{elim}_\mathcal{S}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_\sim \# d \Leftarrow \Pi$

where A_\sim is understood as in Definition 4.2(1).

Proof

We separately prove each implication.

[1. \Rightarrow 2.] (*the transformation is complete*). Assume that T is a $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree witnessing $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We want to show the existence of a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree T' witnessing $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \# d \Leftarrow \Pi$. We reason by complete induction on $\|T\|$. There are three possible cases according to the syntactic form of the atom A . In each case we argue how to build the desired proof tree T' .

- A is a primitive atom κ . In this case A_{\sim} is also κ and T contains only one **SQPA** inference node. Because of the inference rules **SQPA** and **QPA**, both $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ and $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ are equivalent to $\Pi \models_{\mathcal{C}} \kappa$, therefore T' trivially contains just one **QPA** inference node.

- A is an equation $t == s$. In this case A_{\sim} is $t \sim s$ and T contains just one **SQEA** inference node. We know $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} (t == s) \# d \Leftarrow \Pi$ is equivalent to $t \approx_{d, \Pi} s$ because of the inference rule **SQEA**. From this equivalence follows $\text{EQ}_{\mathcal{S}} \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ due to Lemma 4.1 and hence $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ by construction of $\text{elim}_{\mathcal{S}}(\mathcal{P})$. In this case, T' will be a proof tree rooted by a **QDA** inference step.

- A is a defined atom $p'(\bar{t}'_n)$ with $p' \in DP^n$. In this case A_{\sim} is $p'(\bar{t}'_n)$ and the root inference of T must be a **SQDA** inference step of the form:

$$\frac{(\ (t'_i == t_i \theta) \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi \)_{j=1 \dots m}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\clubsuit)$$

with $C : (p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}$, θ substitution, $\mathcal{S}(p', p) = d_0 \neq \mathbf{b}$, $e_j \triangleright^? w_j$ ($1 \leq j \leq m$), $d \trianglelefteq d_i$ ($0 \leq i \leq n$) and $d \trianglelefteq \alpha \circ e_j$ ($1 \leq j \leq m$)—which means $d \trianglelefteq \alpha$ in the case $m = 0$. We can assume that the first n premises at (\clubsuit) are proved in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ w.r.t. \mathcal{P} by proof trees T_{1i} ($1 \leq i \leq n$) satisfying $\|T_{1i}\| < \|T\|$ ($1 \leq i \leq n$), and the last m premises at (\clubsuit) are proved in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ w.r.t. \mathcal{P} by proof trees T_{2j} ($1 \leq j \leq m$) satisfying $\|T_{2j}\| < \|T\|$ ($1 \leq j \leq m$).

By Definition 4.2, we know that the transformed program $\text{elim}_{\mathcal{S}}(\mathcal{P})$ contains two clauses of the following form:

$$\begin{aligned} \hat{C} : \quad & \hat{p}_C(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m \\ \hat{C}_{p'} : \quad & p'(\bar{X}_n) \xleftarrow{\mathbf{t}} \text{pay}_{d_0} \#?, \ (X_i \sim t_i) \#? \)_{i=1 \dots n}, \ \hat{p}_C(\bar{t}_n) \#? \end{aligned}$$

where X_i ($1 \leq i \leq n$) are fresh variables not occurring in C and B_j^i ($1 \leq j \leq m$) is the result of replacing ' \sim ' for ' $==$ ' if B_j is equation; and B_j itself otherwise. Given that the n variables X_i do not occur in C , we can assume that $\sigma =_{\text{def}} \theta' \uplus \theta$ with $\theta' =_{\text{def}} \{X_1 \mapsto t'_1, \dots, X_n \mapsto t'_n\}$ is a well-defined substitution. We claim that $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \# d \Leftarrow \Pi$ can be proved with a proof tree T' rooted by the **QDA** inference step $(\spadesuit.1)$, which uses the clause $\hat{C}_{p'}$ instantiated by σ and having $d_{n+1} = d$.

$$\begin{array}{c} \frac{\begin{array}{l} (\ (t'_i == X_i \sigma) \# \mathbf{t} \Leftarrow \Pi \)_{i=1 \dots n} \\ \text{pay}_{d_0} \sigma \# d_0 \Leftarrow \Pi \\ (\ (X_i \sim t_i) \sigma \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \\ \hat{p}_C(\bar{t}_n) \sigma \# d_{n+1} \Leftarrow \Pi \end{array}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\spadesuit.1) \end{array} \quad \begin{array}{c} \frac{\begin{array}{l} (\ (t'_i == X_i \theta') \# \mathbf{t} \Leftarrow \Pi \)_{i=1 \dots n} \\ \text{pay}_{d_0} \# d_0 \Leftarrow \Pi \\ (\ (X_i \theta' \sim t_i \theta) \# d_i \Leftarrow \Pi \)_{i=1 \dots n} \\ \hat{p}_C(\bar{t}_n \theta) \# d_{n+1} \Leftarrow \Pi \end{array}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\spadesuit.2) \end{array}$$

By construction of σ , $(\spadesuit.1)$ can be rewritten as $(\spadesuit.2)$, and in order to build the rest of T' , we show that each premise of $(\spadesuit.2)$ admits a proof in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ w.r.t. the transformed program $\text{elim}_S(\mathcal{P})$:

- $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t'_i == X_i \theta') \# \mathbf{t} \Leftarrow \Pi$ for $i = 1 \dots n$. Straightforward using a single **QEA** inference step since $X_i \theta' = t'_i$ and $t'_i \approx_{\Pi} t'_i$ is trivially true.
- $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \text{pay}_{d_0} \# d_0 \Leftarrow \Pi$. Immediate using the clause $(\text{pay}_{d_0} \xleftarrow{d_0}) \in \text{elim}_S(\mathcal{P})$ with a single **QDA** inference step.
- $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (X_i \theta' \sim t_i \theta) \# d_i \Leftarrow \Pi$ for $i = 1 \dots n$. From the first n premises of (\clubsuit) we know $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} (t'_i == t_i \theta) \# d_i \Leftarrow \Pi$ with a proof tree T_{1i} satisfying $\|T_{1i}\| < \|T\|$ for $i = 1 \dots n$. Therefore, for $i = 1 \dots n$, $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t'_i \sim t_i \theta) \# d_i \Leftarrow \Pi$ with some $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree T'_{1i} by inductive hypothesis. Since $(X_i \theta' \sim t_i \theta) = (t'_i \sim t_i \theta)$ for $i = 1 \dots n$, we are done.
- $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \hat{p}_C(\bar{t}_n \theta) \# d \Leftarrow \Pi$. This is proved by a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree with a **QDA** inference step node at its root of the following form:

$$\frac{((t_i \theta == t_i \theta) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j^i \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{\hat{p}_C(\bar{t}_n \theta) \# d \Leftarrow \Pi} (\heartsuit)$$

which uses the program clause \hat{C} instantiated by the substitution θ . Once more, we have to check that the premises can be derived in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ from the transformed program $\text{elim}_S(\mathcal{P})$ and that the side conditions of (\heartsuit) are satisfied:

- The first n premises can be trivially proved using **QEA** inference steps.
- The last m premises can be proved w.r.t. $\text{elim}_S(\mathcal{P})$ with some $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof trees T'_{2j} ($1 \leq j \leq m$) by the inductive hypothesis, since we have premises $(B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}$ at (\clubsuit) that can be proved in $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ w.r.t. \mathcal{P} with proof trees T_{2j} of size $\|T_{2j}\| < \|T\|$ ($1 \leq j \leq m$).
- The side conditions—namely: $e_j \triangleright^? w_j$ ($1 \leq j \leq m$), $d \trianglelefteq d_i$ ($1 \leq i \leq n$) and $d \trianglelefteq \alpha \circ e_j$ ($1 \leq j \leq m$)—trivially hold because they are also satisfied by (\clubsuit) .

Finally, we complete the construction of T' by checking that $(\spadesuit.2)$ satisfies the side conditions of the inference rule **QDA**:

- All threshold values at the body of $\hat{C}_{p'}$ are '?', therefore the first group of side conditions becomes $d_i \triangleright^? ?$ ($0 \leq i \leq n+1$), which are trivially true.
- The second side condition reduces to $d \trianglelefteq \mathbf{t}$, which is also trivially true.
- The third, and last, side condition is $d \trianglelefteq \mathbf{t} \circ d_i$ ($0 \leq i \leq n+1$), or equivalently $d \trianglelefteq d_i$ ($0 \leq i \leq n+1$). In fact, $d \trianglelefteq d_i$ ($0 \leq i \leq n$) holds due to the side conditions in (\clubsuit) , and $d \trianglelefteq d_{n+1}$ holds because $d_{n+1} = d$ by construction of $(\spadesuit.1)$ and $(\spadesuit.2)$.

[2. \Rightarrow 1.] (*the transformation is sound*). Assume that T' is a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree witnessing $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} A_{\sim} \# d \Leftarrow \Pi$. We want to show the existence of a $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We reason by complete induction of $\|T'\|$. There are three possible cases according to the syntactic form of the atom A_{\sim} . In each case we argue how to build the desired proof tree T .

— A_{\sim} is a primitive atom κ . In this case A is also κ and T' contains only one **QPA** inference node. Both $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ are equivalent

to $\Pi \models_C \kappa$ because of the inference rules **QPA** and **SQPA**, therefore T trivially contains just one **SQPA** inference node.

— A_\sim is of the form $t \sim s$. In this case A is $t == s$ and T' is rooted by a **QDA** inference step. From $\text{elim}_S(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$ and by construction of $\text{elim}_S(\mathcal{P})$ we have $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (t \sim s) \# d \Leftarrow \Pi$. By Lemma 4.1 we get $t \approx_{d, \Pi} s$ and, by the definition of the **SQEA** inference step, we can build T as a proof tree with only one **SQEA** inference node proving $\mathcal{P} \vdash_{S, \mathcal{D}, \mathcal{C}} (t == s) \# d \Leftarrow \Pi$.

— A_\sim is a defined atom $p'(\bar{t}_n)$ with $p' \in DP^n$ and $p' \neq \sim$. In this case $A = A_\sim$ and the step at the root of T' must be a **QDA** inference step using a clause $C' \in \text{elim}_S(\mathcal{P})$ with head predicate p' and a substitution θ . Because of Definition 4.2 and the fact that p' is relevant for \mathcal{P} , there must be some clause $C : (p(\bar{t}_n) \Leftarrow^\alpha \bar{B}) \in \mathcal{P}$ such that $\mathcal{S}(p, p') = d_0 \neq \mathbf{b}$, and C' must be of the form:

$$C' : p'(\bar{X}_n) \Leftarrow^t \text{pay}_{d_0} \#?, ((X_i \sim t_i) \#?)_{i=1 \dots n}, \hat{p}_C(\bar{t}_n) \#?$$

where the variables \bar{X}_n do not occur in C . Thus the **QDA** inference step at the root of T' must be of the form:

$$\frac{\begin{array}{l} ((t'_i == X_i \theta) \# d_{1i} \Leftarrow \Pi)_{i=1 \dots n} \\ \text{pay}_{d_0} \theta \# e_{10} \Leftarrow \Pi \\ ((X_i \sim t_i) \theta \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n} \\ \hat{p}_C(\bar{t}_n) \theta \# e_{1(n+1)} \Leftarrow \Pi \end{array}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\spadesuit)$$

and the proof of the last premise must use the only clause for \hat{p}_C introduced in $\text{elim}_S(\mathcal{P})$ according to Definition 4.2, i.e.:

$$\hat{C} : \hat{p}_C(\bar{t}_n) \Leftarrow^\alpha B_\sim^1 \# w_1, \dots, B_\sim^m \# w_m.$$

Therefore, the proof of this premise must be of the form:

$$\frac{((t_i \theta == t_i \theta') \# d_{2i} \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j^j \theta' \# e_{2j} \Leftarrow \Pi)_{j=1 \dots m}}{\hat{p}_C(\bar{t}_n) \theta \# e_{1(n+1)} \Leftarrow \Pi} \quad (\heartsuit)$$

for some substitution θ' not affecting \bar{X}_n . We can assume that the last m premises in (\heartsuit) are proved in $\text{QCHL}(\mathcal{D}, \mathcal{C})$ w.r.t. $\text{elim}_S(\mathcal{P})$ by proof trees T'_j satisfying $\|T'_j\| < \|T'\|$ ($1 \leq j \leq m$). Then we use the substitution θ' and clause C to build a $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree T with a **SQDA** inference step at the root of the form:

$$\frac{((t'_i == t_i \theta') \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta' \# e_{2j} \Leftarrow \Pi)_{j=1 \dots m}}{p'(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\clubsuit)$$

Next we check that the premises of this inference step admit proofs in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ and that (\clubsuit) satisfies the side conditions of a valid **SQDA** inference step.

- $\mathcal{P} \vdash_{S, \mathcal{D}, \mathcal{C}} (t'_i == t_i \theta') \# e_{1i} \Leftarrow \Pi$ for $i = 1 \dots n$.
 - From the premises $((X_i \sim t_i) \theta \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n}$ of (\spadesuit) and by construction of $\text{elim}_S(\mathcal{P})$ we know $EQ_S \vdash_{\mathcal{D}, \mathcal{C}} (X_i \sim t_i) \theta \# e_{1i} \Leftarrow \Pi$ ($1 \leq i \leq n$). Therefore by Lemma 4.1 we have $X_i \theta \approx_{e_{1i}, \Pi} t_i \theta$ for $i = 1 \dots n$.
 - Consider now the premises $((t'_i == X_i \theta) \# d_{1i} \Leftarrow \Pi)_{i=1 \dots n}$ of (\spadesuit) . Their proofs

must rely on **QEA** inference steps, and therefore $t'_i \approx_{\Pi} X_i\theta$ holds for $i = 1 \dots n$.

- Analogously, from the proofs of the premises $((t_i\theta == t_i\theta') \# d_{2i} \Leftarrow \Pi)_{i=1 \dots n}$ we have $t_i\theta \approx_{\Pi} t_i\theta'$ (or equivalently $t_i\theta' \approx_{\Pi} t_i\theta$) for $i = 1 \dots n$.

From the previous points we have $X_i\theta \approx_{e_{1i}, \Pi} t_i\theta$, $t'_i \approx_{\Pi} X_i\theta$ and $t_i\theta' \approx_{\Pi} t_i\theta$, which by Lemma 2.7(1) of (Rodríguez-Artalejo and Romero-Díaz 2010b) imply $t'_i \approx_{e_{1i}, \Pi} t_i\theta'$ ($1 \leq i \leq n$). Therefore the premises $((t'_i == t_i\theta') \# e_{1i} \Leftarrow \Pi)_{i=1 \dots n}$ can be proven in $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ using a **SQEA** inference step.

- $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} B_j\theta' \# e_{2j} \Leftarrow \Pi$ for $j = 1 \dots m$. We know $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}, \mathcal{C}} B_j^j\theta' \# e_{2j} \Leftarrow \Pi$ with a proof tree T'_j satisfying $\|T'_j\| < \|T'\|$ ($1 \leq j \leq m$) because of (\heartsuit). Therefore we have, by inductive hypothesis, $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} B_j\theta' \# e_{2j} \Leftarrow \Pi$ for some $\text{SQCHL}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree T_j ($1 \leq j \leq m$).
- $\mathcal{S}(p, p') = d_0 \neq \mathbf{b}$. As seen above.
- $e_{2j} \triangleright^? w_j$ for $j = 1 \dots m$. This is a side condition of the **QDA** step in (\heartsuit).
- $d \trianglelefteq e_{1i}$ for $i = 1 \dots n$. Straightforward from the side conditions of (\spadesuit), which include $d \trianglelefteq \mathbf{t} \circ e_{1i}$ for $(0 \leq i \leq n+1)$.
- $d \trianglelefteq \alpha \circ e_{2j}$ for $j = 1 \dots m$. This follows from the side conditions of (\spadesuit) and (\heartsuit), since we have $d \trianglelefteq \mathbf{t} \circ e_{1i}$ for $i = 0 \dots n+1$ (in particular $d \trianglelefteq e_{1(n+1)}$) and $e_{1(n+1)} \trianglelefteq \alpha \circ e_{2j}$ for $j = 1 \dots m$. \square

Finally, the next theorem extends the previous result to goals.

Theorem 4.2

Let G be a goal for a $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} whose atoms are all relevant for \mathcal{P} . Assume $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ and $G' = \text{elim}_{\mathcal{S}}(G)$. Then, $\text{Sol}_{\mathcal{P}}(G) = \text{Sol}_{\mathcal{P}'}(G')$.

Proof

According to the definition of goals in Section 2, and Definition 4.2, G and G' must be of the form $(A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$ and $(A_i^i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m}$, respectively. By Definitions 2.2 and 3.1, both $\text{Sol}_{\mathcal{P}}(G)$ and $\text{Sol}_{\mathcal{P}'}(G')$ are sets of triples $\langle \sigma, \mu, \Pi \rangle$ where σ is a \mathcal{C} -substitution, $\mu : \text{var}(G) \rightarrow D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ (note that $\text{var}(G) = \text{var}(G')$) and Π is a satisfiable finite set of \mathcal{C} -constraints. Moreover:

1. $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$ iff $W_i\mu = d_i \triangleright^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{S}, \mathcal{D}, \mathcal{C}} A_i\sigma \# W_i\mu \Leftarrow \Pi$ ($1 \leq i \leq m$).
2. $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ iff $W_i\mu = d_i \triangleright^? \beta_i$ and $\mathcal{P}' \vdash_{\mathcal{D}, \mathcal{C}} A_i^i\sigma \# W_i\mu \Leftarrow \Pi$ ($1 \leq i \leq m$).

Because of Theorem 4.1, conditions (1) and (2) are equivalent. \square

4.2 Transforming QCLP into CLP

The results presented in this subsection are dependant on the assumption that the qualification domain \mathcal{D} is existentially expressible in the constraint domain \mathcal{C} via an injective mapping $\iota : D_{\mathcal{D}} \setminus \{\mathbf{b}\} \rightarrow C_{\mathcal{C}}$ and two existential \mathcal{C} -constraints of the following form:

$$\begin{aligned} \text{qVal}(X) &= \exists U_1 \dots \exists U_k (B_1 \wedge \dots \wedge B_m) \\ \text{qBound}(X, Y, Z) &= \exists V_1 \dots \exists V_l (C_1 \wedge \dots \wedge C_q) \end{aligned}$$

Our aim is to present semantically correct transformations from $\text{QCLP}(\mathcal{D}, \mathcal{C})$ into $\text{CLP}(\mathcal{C})$, working both for programs and goals. In order to compute with the encodings of \mathcal{D} values in \mathcal{C} , we will use the $\text{CLP}(\mathcal{C})$ -program $E_{\mathcal{D}}$ consisting of the following two clauses:

$$\begin{aligned} qVal(X) &\leftarrow B_1, \dots, B_m \\ qBound(X, Y, Z) &\leftarrow C_1, \dots, C_q \end{aligned}$$

where $qVal \in DP^1$ and $qBound \in DP^3$ do not occur in the $\text{QCLP}(\mathcal{D}, \mathcal{C})$ programs and goals to be transformed.

The lemma stated below is an immediate consequence of Lemma 3.1 and Definition 2.1.

Lemma 4.2

For any satisfiable finite set Π of \mathcal{C} -constraints one has:

1. For any ground term $t \in C_{\mathcal{C}}$:

$$t \in \text{ran}(\iota) \iff \mathbf{qVal}(t) \text{ true in } \mathcal{C} \iff E_{\mathcal{D}} \vdash_{\mathcal{C}} qVal(t) \Leftarrow \Pi$$

2. For any ground terms $r = \iota(x)$, $s = \iota(y)$, $t = \iota(z)$ with $x, y, z \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$:

$$x \triangleleft y \circ z \iff \mathbf{qBound}(r, s, t) \text{ true in } \mathcal{C} \iff E_{\mathcal{D}} \vdash_{\mathcal{C}} qBound(r, s, t) \Leftarrow \Pi$$

The two items above are also valid if $E_{\mathcal{D}}$ is replaced by any $\text{CLP}(\mathcal{C})$ -program including the two clauses in $E_{\mathcal{D}}$ and having no additional occurrences of $qVal$ and $qBound$ at the head of clauses. \square

Now we are ready to define the transformations from $\text{QCLP}(\mathcal{D}, \mathcal{C})$ into $\text{CLP}(\mathcal{C})$.

Definition 4.3

Assume that \mathcal{D} is existentially expressible in \mathcal{C} , and let $\mathbf{qVal}(X)$, $\mathbf{qBound}(X, Y, Z)$ and $E_{\mathcal{D}}$ be as explained above. Assume also a $\text{QCLP}(\mathcal{D}, \mathcal{C})$ -program \mathcal{P} and a $\text{QCLP}(\mathcal{D}, \mathcal{C})$ -goal G for \mathcal{P} without occurrences of the defined predicate symbols $qVal$ and $qBound$. Then:

1. \mathcal{P} is transformed into the $\text{CLP}(\mathcal{C})$ -program $\text{elim}_{\mathcal{D}}(\mathcal{P})$ consisting of the two clauses in $E_{\mathcal{D}}$ and the transformed $C^{\mathcal{T}}$ of each clause $C \in \mathcal{P}$, built as specified in Figure 5. The transformation rules of this figure assume a different choice of $p' \in DP^{n+1}$ for each $p \in DP^n$.
2. G is transformed into the $\text{CLP}(\mathcal{C})$ -goal $\text{elim}_{\mathcal{D}}(G)$ built as specified in Figure 5. Note that the qualification variables \overline{W}_n occurring in G become normal CLP variables in the transformed goal. \square

The following example illustrates the transformation $\text{elim}_{\mathcal{D}}$.

Example 4.3 (Running example: $\text{CLP}(\mathcal{R})$ -program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$)

Consider the $\text{QCLP}(\mathcal{U} \otimes \mathcal{W}, \mathcal{R})$ -program $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ and the goal $\text{elim}_{\mathcal{S}}(G_r)$ for the same program as presented in Example 4.2. The transformed $\text{CLP}(\mathcal{R})$ -program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ is as follows:

Transforming Atoms

$$\mathbf{TEA} \quad (t == s)^T = (t == s, \iota(\mathbf{t})).$$

$$\mathbf{TPA} \quad (\kappa)^T = (\kappa, \iota(\mathbf{t})) \text{ with } \kappa \text{ primitive atom.}$$

$$\mathbf{TDA} \quad (p(\bar{t}_n))^T = (p'(\bar{t}_n, W), W) \text{ with } p \in DP^n \text{ and } W \text{ a fresh CLP variable.}$$

Transforming qc-Atoms

$$\mathbf{TQCA} \quad \frac{A^T = (A', w)}{(A \# d \Leftarrow \Pi)^T = (A' \Leftarrow \Pi, \{qVal(w), qBound(\iota(d), \iota(\mathbf{t}), w)\})}$$

Transforming Program Clauses

$$\mathbf{TPC} \quad \frac{(B_j^T = (B'_j, w'_j))_{j=1\dots m}}{C^T = p'(\bar{t}_n, W) \leftarrow qVal(W), \left(\begin{array}{l} qVal(w'_j), \lceil w'_j \rceil \triangleright^? \iota(w_j)^\neg, \\ qBound(W, \iota(\alpha), w'_j), B'_j \end{array} \right)_{j=1\dots m}}$$

where $C : p(\bar{t}_n) \xleftarrow{\alpha} B_1 \# w_1, \dots, B_m \# w_m$, W is a fresh CLP variable and $\lceil w'_j \rceil \triangleright^? \iota(w_j)^\neg$ is omitted if $w_j = ?$, i.o.c. abbreviates $qBound(\iota(w_j), \iota(\mathbf{t}), w'_j)$.

Transforming Goals

$$\mathbf{TG} \quad \frac{(B_j^T = (B'_j, w'_j))_{j=1\dots m}}{\text{elim}_{\mathcal{D}}(G) = \left(\begin{array}{l} qVal(W_j), \lceil W_j \rceil \triangleright^? \iota(\beta_j)^\neg, \\ qVal(w'_j), qBound(W_j, \iota(\mathbf{t}), w'_j), B'_j \end{array} \right)_{j=1\dots m}}$$

where $G : (B_j \# W_j, W_j \triangleright^? \beta_j)_{j=1\dots m}$ and $\lceil W_j \rceil \triangleright^? \iota(\beta_i)^\neg$ as in **TPC** above.

Fig. 5. Transformation rules

$$\begin{aligned} \hat{R}_1 \quad & famous_{R_1}(sha, W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (0.9, 1)) \\ R_{1.1} \quad & famous(X, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{\mathbf{t}}(W_1), \\ & qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2), \\ & qVal(W_3), qBound(W, \mathbf{t}, W_3), famous_{R_1}(sha, W_3) \\ \hat{R}_2 \quad & wrote_{R_2}(sha, kle, W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (1, 1)) \\ R_{2.1} \quad & wrote(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{\mathbf{t}}(W_1), \\ & qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2), \\ & qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, kle, W_3), \\ & qVal(W_4), qBound(W, \mathbf{t}, W_4), wrote_{R_2}(sha, kle, W_4) \\ R_{2.2} \quad & authored(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{(0.9, 0)}(W_1), \\ & qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2), \\ & qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, kle, W_3), \\ & qVal(W_4), qBound(W, \mathbf{t}, W_4), wrote_{R_2}(sha, kle, W_4) \\ \hat{R}_3 \quad & wrote_{R_3}(sha, hamlet, W) \leftarrow qVal(W), qBound(W, \mathbf{t}, (1, 1)) \\ R_{3.1} \quad & wrote(X, Y, W) \leftarrow qVal(W), qVal(W_1), qBound(W, \mathbf{t}, W_1), pay_{\mathbf{t}}(W_1), \\ & qVal(W_2), qBound(W, \mathbf{t}, W_2), \sim(X, sha, W_2), \\ & qVal(W_3), qBound(W, \mathbf{t}, W_3), \sim(Y, hamlet, W_3), \\ & qVal(W_4), qBound(W, \mathbf{t}, W_4), wrote_{R_3}(sha, hamlet, W_4) \end{aligned}$$

```

R3.2   authored(X, Y, W) ← qVal(W), qVal(W1), qBound(W, t, W1), pay(0.9,0)(W1),
        qVal(W2), qBound(W, t, W2), ∼(X, sha, W2),
        qVal(W3), qBound(W, t, W3), ∼(Y, hamlet, W3),
        qVal(W4), qBound(W, t, W4), wroteR3(sha, hamlet, W4)

R4     good_workR4(X, W) ← qVal(W),
        qVal(W1), qBound((0.5,100), t, W1), qBound(W, (0.75,3), W1), famous(Y, W1),
        qVal(W2), qBound(W, (0.75,3), W2), authored(Y, X, W2)

R4.1   good_work(G, W) ← qVal(W), qVal(W1), qBound(W, t, W1), payt(W1),
        qVal(W2), qBound(W, t, W2), ∼(G, X, W2),
        qVal(W3), qBound(W, t, W3), good_workR4(X, W3)

% Program clauses for ∼:
∼(X, Y, W) ← qVal(W), qVal(t), qBound(W, t, t), X=Y
∼(kle, kli, W) ← qVal(W), qVal(W1), qBound(W, t, W1), pay(0.8,2)(W1)
[... ]

% Program clauses for pay:
payt(W) ← qVal(W), qBound(W, t, t)
pay(0.9,0)(W) ← qVal(W), qBound(W, t, (0.9,0))
pay(0.8,2)(W) ← qVal(W), qBound(W, t, (0.8,2))

% Program clauses for qVal & qBound:
qVal((X1,X2)) ← X1 > 0, X1 ≤ 1, X2 ≥ 0
qBound((W1,W2), (Y1,Y2), (Z1,Z2)) ← W1 ≤ Y1 × Z1, W2 ≥ Y2 + Z2
    
```

Finally, the goal $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G_r))$ for $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ is as follows:

$qVal(W), qBound((0.5,10), t, W), qVal(W'), qBound(W, t, W'), good_work(X, W')$

Note that, in order to improve the clarity of the program clauses of this example, the qualification value (1,0)—top value in $\mathcal{U} \otimes \mathcal{W}$ —has been replaced by **t**. \square

The next theorem proves the semantic correctness of the program transformation.

Theorem 4.3

Let A be an atom such that $qVal$ and $qBound$ do not occur in A . Assume $d \in D \setminus \{\mathbf{b}\}$ such that $(A \# d \Leftarrow \Pi)^{\mathcal{T}} = (A' \Leftarrow \Pi, \Omega)$. Then, the two following statements are equivalent:

1. $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$
2. $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A' \rho \Leftarrow \Pi$ for some $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ such that $\text{dom}(\rho) = \text{var}(\Omega)$.

Proof

We separately prove each implication.

[1. \Rightarrow 2.] (*the transformation is complete*). We assume that T is a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree witnessing $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We want to show the existence of a $\text{CLP}(\mathcal{C})$ proof tree T' witnessing $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A' \rho \Leftarrow \Pi$ for some $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ such that $\text{dom}(\rho) = \text{var}(\Omega)$. We reason by complete induction on $\|T\|$. There are three possible

cases, according to the syntactic form of the atom A . In each case we argue how to build the desired proof tree T' .

— A is a primitive atom κ . In this case **TQCA** and **TPA** compute $A' = \kappa$ and $\Omega = \{\mathbf{qVal}(\iota(\mathbf{t})), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Now, from $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ follows $\Pi \models_{\mathcal{C}} \kappa$ due to the **QPA** inference, and therefore taking $\rho = \varepsilon$ we can prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \kappa \varepsilon \Leftarrow \Pi$ with a proof tree T' containing only one **PA** node. Moreover, $\varepsilon \in \text{Sol}_{\mathcal{C}}(\Omega)$ is trivially true because the two constraints belonging to Ω are obviously true in \mathcal{C} .

— A is an equation $t == s$. In this case **TQCA** and **TEA** compute $A' = (t == s)$ and $\Omega = \{\mathbf{qVal}(\iota(\mathbf{t})), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Now, from $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} (t == s) \# d \Leftarrow \Pi$ follows $t \approx_{\Pi} s$ due to the **QEA** inference, and therefore taking $\rho = \varepsilon$ we can prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t == s) \varepsilon \Leftarrow \Pi$ with a proof tree T' containing only one **EA** node. Moreover, $\varepsilon \in \text{Sol}_{\mathcal{C}}(\Omega)$ is trivially true because the two constraints belonging to Ω are obviously true in \mathcal{C} .

— A is a defined atom $p(\bar{t}'_n)$ with $p \in DP^n$. In this case **TQCA** and **TDA** compute $A' = p'(\bar{t}'_n, W)$ and $\Omega = \{\mathbf{qVal}(W), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), W)\}$ where W is a fresh CLP variable. On the other hand, T must be rooted by a **QDA** step of the form:

$$\frac{(\iota'_i == t_i \theta) \# d_i \Leftarrow \Pi)_{i=1 \dots n} \quad (B_j \theta \# e_j \Leftarrow \Pi)_{j=1 \dots m}}{p(\bar{t}'_n) \# d \Leftarrow \Pi} \quad (\clubsuit)$$

using a clause $C : (p(\bar{t}_n) \Leftarrow^\alpha B_1 \# w_1, \dots, B_m \# w_m) \in \mathcal{P}$ instantiated by a substitution θ and such that the side conditions $e_j \triangleright^? w_j$ ($1 \leq j \leq m$), $d \leq d_i$ ($1 \leq i \leq n$) and $d \leq \alpha \circ e_j$ ($1 \leq j \leq m$) are fulfilled.

For $j = 1 \dots m$ we can assume $B_j^T = (B'_j, w'_j)$ and thus $(B_j \theta \# e_j \Leftarrow \Pi)^T = (B'_j \theta \Leftarrow \Pi, \Omega_j)$ where $\Omega_j = \{\mathbf{qVal}(w'_j), \mathbf{qBound}(\iota(e_j), \iota(\mathbf{t}), w'_j)\}$. The proof trees T_j of the last m premises of (\clubsuit) will have less than $\|T\|$ nodes, and hence the induction hypothesis can be applied to each $(B_j \theta \# e_j \Leftarrow \Pi)$ with $1 \leq j \leq m$, obtaining $\text{CHL}(\mathcal{C})$ proof trees T'_j proving $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta \rho_j \Leftarrow \Pi$ for some $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ with $\text{dom}(\rho_j) = \text{var}(\Omega_j)$.

Consider $\rho = \{W \mapsto \iota(d)\}$ and $C^T \in \text{elim}_{\mathcal{D}}(\mathcal{P})$ of the form:

$$C^T : p'(\bar{t}'_n, W') \Leftarrow \mathbf{qVal}(W'), \left(\begin{array}{l} \mathbf{qVal}(w'_j), \ulcorner w'_j \triangleright^? \iota(w_j) \urcorner, \\ \mathbf{qBound}(W', \iota(\alpha), w'_j), B'_j \end{array} \right)_{j=1 \dots m}.$$

Obviously, $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ and $\text{dom}(\rho) = \text{var}(\Omega)$. To finish the proof we must prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A' \rho \Leftarrow \Pi$. We claim that this can be done with a $\text{CHL}(\mathcal{C})$ proof tree T' whose root inference is a **DA** step of the form:

$$\frac{\begin{array}{l} (\iota'_i \rho == t_i \theta') \Leftarrow \Pi)_{i=1 \dots n} \\ (W \rho == W' \theta') \Leftarrow \Pi \\ \mathbf{qVal}(W') \theta' \Leftarrow \Pi \\ \left(\begin{array}{l} \mathbf{qVal}(w'_j) \theta' \Leftarrow \Pi \\ \ulcorner w'_j \triangleright^? \iota(w_j) \urcorner \theta' \Leftarrow \Pi \\ \mathbf{qBound}(W', \iota(\alpha), w'_j) \theta' \Leftarrow \Pi \\ B'_j \theta' \Leftarrow \Pi \end{array} \right)_{j=1 \dots m} \end{array}}{p'(\bar{t}'_n, W) \rho \Leftarrow \Pi} \quad (\spadesuit)$$

using \mathcal{C}^T instantiated by the substitution $\theta' = \theta \uplus \rho_1 \uplus \dots \uplus \rho_m \uplus \{W' \mapsto \iota(d)\}$. We check that the premises of (\spadesuit) can be derived from $\text{elim}_{\mathcal{D}}(\mathcal{P})$ in $\text{CHL}(\mathcal{C})$:

- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t'_i \rho == t_i \theta') \Leftarrow \Pi$ for $i = 1 \dots n$. By construction of ρ and θ' , these are equivalent to prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t'_i == t_i \theta) \Leftarrow \Pi$ for $i = 1 \dots n$ and these hold with $\text{CHL}(\mathcal{C})$ proof trees of only one **EA** node because of $t'_i \approx_{\Pi} t_i \theta$, which is a consequence of the first n premises of (\clubsuit) .
- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (W \rho == W' \theta') \Leftarrow \Pi$. By construction of ρ and θ' , this is equivalent to prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (\iota(d) == \iota(d)) \Leftarrow \Pi$ which results trivial.
- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \text{qVal}(W') \theta' \Leftarrow \Pi$. By construction of θ' , this is equivalent to prove $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \text{qVal}(\iota(d)) \Leftarrow \Pi$. We trivially have that $\iota(d) \in \text{ran}(\iota)$. Then, by Lemma 4.2, this premise holds.
- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \text{qVal}(w'_j) \theta' \Leftarrow \Pi$ for $j = 1 \dots m$. By construction of θ' and Lemma 4.2 we must prove, for any fixed j , that $\text{qVal}(w'_j \rho_j)$ is true in \mathcal{C} . As $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ we know $\rho_j \in \text{Sol}_{\mathcal{C}}(\text{qVal}(w'_j))$, therefore $\text{qVal}(w'_j \rho_j)$ is trivially true in \mathcal{C} .
- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \ulcorner w'_j \triangleright^? \iota(w_j) \urcorner \theta' \Leftarrow \Pi$ for $j = 1 \dots m$. We reason for any fixed j . If $w_j = ?$ this results trivial. Otherwise, it amounts to $\text{qBound}(\iota(w_j), \iota(\mathbf{t}), w'_j \rho_j)$ being true in \mathcal{C} , by construction of θ' and Lemma 4.2. As seen before, $\text{qVal}(w'_j \rho_j)$ is true in \mathcal{C} , therefore $w'_j \rho_j = \iota(e'_j)$ for some $e'_j \in D \setminus \{\mathbf{b}\}$. From the side conditions of (\clubsuit) we have $w_j \trianglelefteq e_j$. On the other hand, $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ and, in particular, $\rho_j \in \text{Sol}_{\mathcal{C}}(\text{qBound}(\iota(e_j), \iota(\mathbf{t}), w'_j))$. This, together with $w'_j \rho_j = \iota(e'_j)$, means $e_j \trianglelefteq e'_j$, which with $w_j \trianglelefteq e_j$ implies $w_j \trianglelefteq e'_j$, i.e. $\text{qBound}(\iota(w_j), \iota(\mathbf{t}), w'_j \rho_j)$ is true in \mathcal{C} .
- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \text{qBound}(W', \iota(\alpha), w'_j) \theta' \Leftarrow \Pi$ for $j = 1 \dots m$. We reason for any fixed j . By construction of θ' and Lemma 4.2, we must prove that $\text{qBound}(\iota(d), \iota(\alpha), w'_j \rho_j)$ is true in \mathcal{C} . As seen before, $\text{qVal}(w'_j \rho_j)$ is true in \mathcal{C} , therefore $w'_j \rho_j = \iota(e'_j)$ for some $e'_j \in D \setminus \{\mathbf{b}\}$. From the side conditions of (\clubsuit) we have $d \trianglelefteq \alpha \circ e_j$. On the other hand, $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ and, in particular, $\rho_j \in \text{Sol}_{\mathcal{C}}(\text{qBound}(\iota(e_j), \iota(\mathbf{t}), w'_j))$. This, together with $w'_j \rho_j = \iota(e'_j)$, means $e_j \trianglelefteq e'_j$. Now, $d \trianglelefteq \alpha \circ e_j$ and $e_j \trianglelefteq e'_j$ implies $d \trianglelefteq \alpha \circ e'_j$, i.e. $\text{qBound}(\iota(d), \iota(\alpha), w'_j \rho_j)$ is true in \mathcal{C} .
- $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta' \Leftarrow \Pi$ for $j = 1 \dots m$. In this case, it is easy to see that $B'_j \theta' = B'_j \theta \rho_j$ by construction of θ' and because of the program transformation rules. On the other hand, proof trees T'_j proving $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j \theta \rho_j \Leftarrow \Pi$ can be obtained by inductive hypothesis as seen before.

[2. \Rightarrow 1.] (*the transformation is sound*). We assume that T' is a $\text{CHL}(\mathcal{C})$ proof tree witnessing $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A' \rho \Leftarrow \Pi$ for some $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ such that $\text{dom}(\rho) = \text{var}(\Omega)$. We want to show the existence of a $\text{QCHL}(\mathcal{D}, \mathcal{C})$ proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A \# d \Leftarrow \Pi$. We reason by complete induction on $\|T'\|$. There are three possible cases according to the syntactic form of the atom A' . In each case we argue how to build the desired proof tree T .

— A' is a primitive atom κ . In this case due to **TQCA** and **TPA** we can assume $A = \kappa$ and $\Omega = \{\text{qVal}(\iota(\mathbf{t})), \text{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Note that $\text{dom}(\rho) = \text{var}(\Omega) = \emptyset$ implies $\rho = \varepsilon$. Now, from $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \kappa \varepsilon \Leftarrow \Pi$ follows $\Pi \models_{\mathcal{C}} \kappa$ due to the **PA** inference, and therefore we can prove $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} \kappa \# d \Leftarrow \Pi$ with a proof tree T containing only one **QPA** node.

— A' is an equation $t == s$. In this case due to **TQCA** and **TEA** we can assume

$A = (t == s)$ and $\Omega = \{\mathbf{qVal}(\iota(\mathbf{t})), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), \iota(\mathbf{t}))\}$. Note that $\text{dom}(\rho) = \text{var}(\Omega) = \emptyset$ implies $\rho = \varepsilon$. Now, from $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t == s)\varepsilon \Leftarrow \Pi$ follows $t \approx_{\Pi} s$ due to the **EA** inference, and therefore we can prove $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} (t == s)\sharp d \Leftarrow \Pi$ with a proof tree T containing only one **QEA** node.

— A' is a defined atom $p'(\bar{t}'_n, W)$ with $p' \in DP^{n+1}$. In this case due to **TQCA** and **TDA** we can assume $A = p(\bar{t}'_n)$ and $\Omega = \{\mathbf{qVal}(W), \mathbf{qBound}(\iota(d), \iota(\mathbf{t}), W)\}$. On the other hand, T' must be rooted by a **DA** step (\spadesuit) using a clause $C^{\mathcal{T}} \in \text{elim}_{\mathcal{D}}(\mathcal{P})$ instantiated by a substitution θ' . We can assume that (\spadesuit), $C^{\mathcal{T}}$ and the corresponding clause $C \in \mathcal{P}$ have the form already displayed in [1. \Rightarrow 2.].

By construction of $C^{\mathcal{T}}$, we can assume $B_j^{\mathcal{T}} = (B'_j, w'_j)$. Let $\theta = \theta' \upharpoonright \text{var}(C)$ and $\rho_j = \theta' \upharpoonright \text{var}(w'_j)$ ($1 \geq j \geq m$). Then, due to the premises $\mathbf{qVal}(w'_j)\theta' \Leftarrow \Pi$ of (\spadesuit) and Lemma 4.2 we can assume $e'_j \in D \setminus \{\mathbf{b}\}$ ($1 \leq j \leq m$) such that $w'_j\rho_j = \iota(e'_j)$.

To finish the proof, we must prove $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} A\sharp d \Leftarrow \Pi$. We claim that this can be done with a **QCHL**(\mathcal{D}, \mathcal{C}) proof tree T whose root inference is a **QDA** step of the form of (\clubsuit), as displayed in [1. \Rightarrow 2.], using clause C instantiated by θ . In the premises of this inference we choose $d_i = \mathbf{t}$ ($1 \leq i \leq n$) and $e_j = e'_j$ ($1 \leq j \leq m$). Next we check that these premises can be derived from \mathcal{P} in **QCHL**(\mathcal{D}, \mathcal{C}) and that the side conditions are fulfilled:

- $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \dots n$. This amounts to $t'_i \approx_{\Pi} t_i\theta$ which follows from the first n premises of (\spadesuit) given that $t'_i\rho = t'_i$ and $t_i\theta' = t_i\theta$.
- $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j\theta\sharp e_j \Leftarrow \Pi$ for $j = 1 \dots m$. From $B_j^{\mathcal{T}} = (B'_j, w'_j)$ and due to rule **TQCA**, we have $((B_j\theta)\sharp e_j \Leftarrow \Pi)^{\mathcal{T}} = (B_j\theta \Leftarrow \Pi, \Omega_j)$ where $\Omega_j = \{\mathbf{qVal}(w'_j), \mathbf{qBound}(\iota(e_j), \iota(\mathbf{t}), w'_j)\}$. From the premises of (\spadesuit) and the fact that $B'_j\theta' = B'_j\theta\rho_j$ we know that $\text{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j\theta\rho_j \Leftarrow \Pi$ with a **CHL**(\mathcal{C}) proof tree T'_j such that $\|T'_j\| < \|T'\|$. Therefore $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j\theta\sharp e_j \Leftarrow \Pi$ follows by inductive hypothesis provided that $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$. In fact, due to the form of Ω_j , $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ holds iff $w'_j\rho_j = \iota(e'_j)$ for some e'_j such that $e_j \trianglelefteq e'_j$, which is the case because of the choice of e_j .
- $e_j \triangleright^? w_j$ for $j = 1 \dots m$. Trivial in the case that $w_j = ?$. Otherwise they are equivalent to $w_j \trianglelefteq e'_j$ which follow from premises $\ulcorner w'_j \triangleright^? \iota(w_j) \urcorner \theta' \Leftarrow \Pi$ (i.e. $\ulcorner w'_j\rho_j \triangleright^? \iota(w_j) \urcorner \Leftarrow \Pi$) of (\spadesuit) and Lemma 4.2.
- $d \trianglelefteq d_i$ for $i = 1 \dots n$. Trivially hold due to the choice of $d_i = \mathbf{t}$.
- $d \trianglelefteq \alpha \circ e_j$ for $j = 1 \dots m$. Note that $\rho \in \text{Sol}_{\mathcal{C}}(\Omega)$ implies the existence of $d' \in D \setminus \{\mathbf{b}\}$ such that $\iota(d') = W\rho$ and $d \trianglelefteq d'$. On the other hand, $e_j = e'_j$ by choice. It suffices to prove $d' \trianglelefteq \alpha \circ e'_j$ for $j = 1 \dots m$. Premises of (\spadesuit) and Lemma 4.2 imply that $\mathbf{qBound}(W'\theta', \iota(\alpha), w'_j\theta')$ is true in \mathcal{C} . Moreover, $W'\theta' = W\rho = \iota(d')$ because of another premise of (\spadesuit) and $w'_j\theta' = \iota(e'_j)$ as explained above. Therefore $\mathbf{qBound}(W'\theta', \iota(\alpha), w'_j\theta')$ amounts to $\mathbf{qBound}(\iota(d'), \iota(\alpha), \iota(e'_j))$ which guarantees $d' \trianglelefteq \alpha \circ e'_j$ ($1 \leq j \leq m$). \square

The goal transformation correctness is established by the next theorem, which will rely on the previous result:

Theorem 4.4

Let G be a goal for a **QCLP**(\mathcal{D}, \mathcal{C})-program \mathcal{P} such that \mathbf{qVal} and \mathbf{qBound} do not occur in G . Let $\mathcal{P}' = \text{elim}_{\mathcal{D}}(\mathcal{P})$ and $G' = \text{elim}_{\mathcal{D}}(G)$. Assume a \mathcal{C} -substitution σ ,

a mapping $\mu : \text{var}(G) \rightarrow D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ and a satisfiable finite set of \mathcal{C} -constraints Π . Then, the following two statements are equivalent:

1. $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$.
2. $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ for some θ that verifies the following requirements:
 - (a) $\theta =_{\text{var}(G)} \sigma$,
 - (b) $\theta =_{\text{var}(G)} \mu$ and
 - (c) $W\theta \in \text{ran}(\iota)$ for each $W \in \text{var}(G') \setminus (\text{var}(G) \cup \text{war}(G))$.

Proof

As explained in Subsection 3.1 the syntax of goals in QCLP(\mathcal{D}, \mathcal{C})-programs is the same as that of goals for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs, which is described in Section 2. Therefore G , and G' due to rule **TG**, must have the following form:

$$\begin{aligned} G &: (B_j \# W_j, W_j \triangleright^? \beta_j)_{j=1 \dots m} \\ G' &: (qVal(W_j), \ulcorner W_j \triangleright^? \iota(\beta_j) \urcorner, qVal(w'_j), qBound(W_j, \iota(\mathbf{t}), w'_j), B'_j)_{j=1 \dots m} \end{aligned}$$

with $B_j^T = (B'_j, w'_j)$ ($1 \leq j \leq m$). Note that, because of rule **TQCA**, we have $(B_j \sigma \# W_j \mu \Leftarrow \Pi)^T = (B'_j \sigma \Leftarrow \Pi, \Omega_j)$ with $\Omega_j = \{qVal(w'_j), qBound(\iota(W_j \mu), \iota(\mathbf{t}), w'_j)\}$ for $j = 1 \dots m$. We now prove each implication.

[1. \Rightarrow 2.] Let $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$. This means, by Definition 3.1, $W_j \mu \triangleright^? \beta_j$ and $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j \sigma \# W_j \mu \Leftarrow \Pi$ for $j = 1 \dots m$. In these conditions, Theorem 4.3 guarantees $\mathcal{P}' \vdash_{\mathcal{C}} B'_j \sigma \rho_j \Leftarrow \Pi$ ($1 \leq j \leq m$) for some $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ such that $\text{dom}(\rho_j) = \text{var}(\Omega_j)$. It is easy to see that $\text{var}(G') \setminus (\text{var}(G) \cup \text{war}(G)) = \text{var}(\Omega_1) \uplus \dots \uplus \text{var}(\Omega_m)$. Therefore it is possible to define a substitution θ verifying $\theta =_{\text{var}(G)} \sigma$, $\theta =_{\text{var}(G)} \mu$ and $\theta =_{\text{dom}(\rho_j)} \rho_j$ ($1 \leq j \leq m$). Trivially, θ satisfies conditions 2.(a) and 2.(b). It also satisfies condition 2.(c) because for any j and any variable X such that $X \in \text{var}(\Omega_j)$, we have a constraint $qVal(X) \in \Omega_j$ implying, due to Lemma 4.2, $X\rho_j \in \text{ran}(\iota)$ (because $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$).

In order to prove $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ in the sense of Definition 3.2 we check the following items:

- By construction, θ is a \mathcal{C} -substitution.
- By the theorem's assumptions, Π is a satisfiable and finite set of \mathcal{C} -constraints.
- $\mathcal{P}' \vdash_{\mathcal{C}} A\theta \Leftarrow \Pi$ for every atom A in G' . Because of the form of G' we have to prove the following for any fixed j :
 - $\mathcal{P}' \vdash_{\mathcal{C}} qVal(W_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $qVal(\iota(W_j \mu))$ being true in \mathcal{C} , which is trivial consequence of $W_j \mu \in D \setminus \{\mathbf{b}\}$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} \ulcorner W_j \triangleright^? \iota(\beta_j) \urcorner \theta \Leftarrow \Pi$. If $\beta_j = ?$ this becomes trivial. Otherwise, $W_j \theta = \iota(W_j \mu)$ by construction of θ , and by Lemma 4.2 it suffices to prove $qBound(\iota(\beta_j), \iota(\mathbf{t}), \iota(W_j \mu))$ is true in \mathcal{C} . This follows from $W_j \mu \triangleright^? \beta_j$, that is ensured by $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} qVal(w'_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $qVal(w'_j \rho_j)$ being true in \mathcal{C} , that is guaranteed by $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$.
 - $\mathcal{P}' \vdash_{\mathcal{C}} qBound(W_j, \iota(\mathbf{t}), w'_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $qBound(\iota(W_j \mu), \iota(\mathbf{t}), w'_j \rho_j)$ being true in \mathcal{C} , that is also guaranteed by $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$.

— $\mathcal{P}' \vdash_{\mathcal{C}} B'_j \theta \Leftarrow \Pi$. Note that, by construction of θ , $B'_j \theta = B'_j \sigma \rho_j$. On the other hand, ρ_j has been chosen above to verify $\mathcal{P}' \vdash_{\mathcal{C}} B'_j \sigma \rho_j \Leftarrow \Pi$.

[2. \Rightarrow 1.] Let $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ and assume that θ verifies 2.(a), 2.(b) and 2.(c). In order to prove $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$ in the sense of Definition 3.1 we must prove the following items:

- By the theorem's assumptions, σ is a \mathcal{C} -substitution, $\mu : \text{var}(G) \rightarrow D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ and Π is a satisfiable finite set of \mathcal{C} -constraints.
- $W_j \mu \triangleright^? \beta_j$. We reason for any fixed j . If $\beta_j = ?$ this results trivial. Otherwise, we have $\mathcal{P}' \vdash_{\mathcal{C}} \ulcorner W_j \triangleright^? \iota(\beta_j) \urcorner \theta \Leftarrow \Pi$ which, by condition 2.(b) and Lemma 4.2 amounts to $\text{qBound}(\iota(\beta_j), \iota(\mathbf{t}), \iota(W_j \mu))$ is true \mathcal{C} , i.e. $W_j \mu \triangleright \beta_j$.
- $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j \sigma \# W_j \mu \Leftarrow \Pi$ for $j = 1 \dots m$. We reason for any fixed j . Let ρ_j be the restriction of θ to $\text{var}(\Omega_j)$. Then, $\mathcal{P}' \vdash_{\mathcal{C}} B'_j \sigma \rho_j \Leftarrow \Pi$ follows from $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ and $B'_j \theta = B'_j \sigma \rho_j$. Therefore, $\mathcal{P} \vdash_{\mathcal{D}, \mathcal{C}} B_j \sigma \# W_j \mu \Leftarrow \Pi$ follows from Theorem 5.3 provided that $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$. By Lemma 4.2 and the form of Ω_j , $\rho_j \in \text{Sol}_{\mathcal{C}}(\Omega_j)$ holds iff $\mathcal{P}' \vdash_{\mathcal{C}} \text{qVal}(w'_j \rho_j) \Leftarrow \Pi$ and $\mathcal{P}' \vdash_{\mathcal{C}} \text{qBound}(\iota(W_j \mu), \iota(\mathbf{t}), w'_j \rho_j) \Leftarrow \Pi$, which is true because $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$ and construction of ρ_j . \square

4.3 Solving SQCLP Goals

In this subsection we show that the transformations from the two previous subsections can be used to define abstract goal solving systems for SQCLP and arguing about their correctness. In the sequel we consider a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} and a goal G for \mathcal{P} whose atoms are all relevant for \mathcal{P} . We also consider $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$, $G' = \text{elim}_{\mathcal{S}}(G)$, $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$ and $G'' = \text{elim}_{\mathcal{D}}(G')$. Due to the definition of both $\text{elim}_{\mathcal{S}}$ and $\text{elim}_{\mathcal{D}}$, we can assume:

$$\begin{aligned} G &: (A_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m} \\ G' &: (A'_i \# W_i, W_i \triangleright^? \beta_i)_{i=1 \dots m} \\ G'' &: (\text{qVal}(W_i), \ulcorner W_i \triangleright^? \iota(\beta_i) \urcorner, \text{qVal}(w'_i), \text{qBound}(W_i, \iota(\mathbf{t}), w'_i), A'_i)_{i=1 \dots m} \\ &\quad \text{where } A'_i{}^T = (A'_i, w'_i). \end{aligned}$$

We start by presenting an auxiliary result.

Lemma 4.3

Assume \mathcal{P} , G , \mathcal{P}' , G' , \mathcal{P}'' and G'' as above. Let $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$, $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ and $\theta = \sigma' \nu$. Then $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$. Moreover, $W \theta \in \text{ran}(\iota)$ for every $W \in \text{var}(G'') \setminus \text{var}(G)$.¹

Proof

Consider an arbitrary atom A'' occurring in G'' . Because of $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ we have $\mathcal{P} \vdash_{\mathcal{C}} A'' \sigma' \Leftarrow \Pi$. On the other hand, because of $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ we have $\emptyset \models_{\mathcal{C}} \Pi \nu$ and therefore also $\Pi \models_{\mathcal{C}} \Pi \nu$. This and Definition 3.1(4) of (Rodríguez-Artalejo and Romero-Díaz 2010b) ensure $A'' \sigma' \Leftarrow \Pi \succ_{\mathcal{C}} A'' \sigma' \nu \Leftarrow \Pi$, i.e. $A'' \sigma' \Leftarrow \Pi \succ_{\mathcal{C}} A'' \theta \Leftarrow \Pi$.

¹ Note that $\text{var}(G) \subseteq \text{var}(G'') \setminus \text{var}(G)$.

This fact, $\mathcal{P}'' \vdash_{\mathcal{C}} A''\sigma' \Leftarrow \Pi$ and the Entailment Property for Programs in $\text{CLP}(\mathcal{C})$ imply $\mathcal{P}'' \vdash_{\mathcal{C}} A''\theta \Leftarrow \Pi$. Therefore, $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$.

Consider now any $W \in \text{var}(G'') \setminus \text{var}(G)$. By construction of G'' , one of the atoms occurring in G'' is $q\text{Val}(W)$. Then, due to $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ we have $\mathcal{P}'' \vdash_{\mathcal{C}} q\text{Val}(W\sigma') \Leftarrow \Pi$. Because of Lemma 3.1(1) this implies $\Pi \models_{\mathcal{C}} q\text{Val}(W\sigma')$, i.e. $\text{Sol}_{\mathcal{C}}(\Pi) \subseteq \text{Sol}_{\mathcal{C}}(q\text{Val}(W\sigma'))$. Since $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ we get $\nu \in \text{Sol}_{\mathcal{C}}(q\text{Val}(W\sigma'))$, i.e. $W\sigma'\nu \in \text{ran}(\iota)$. Since $W\sigma'\nu = W\theta$, we are done. \square

Next, we explain how to define an abstract goal solving system for SQCLP from a given abstract goal solving system for CLP.

Definition 4.4

Let CLP-AGSS be an abstract goal solving system for $\text{CLP}(\mathcal{C})$ (in the sense of Definition 3.3). Then we define *SQCLP-AGSS* as an abstract goal solving system for $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ that works as follows:

1. Given a goal G for the $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ -program \mathcal{P} , consider \mathcal{P}' , G' , \mathcal{P}'' and G'' as explained at the beginning of the subsection.
2. For each solution $\langle \sigma', \Pi \rangle$ computed by CLP-AGSS for G'' , \mathcal{P}'' and for any $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$, SQCLP-AGSS computes $\langle \sigma, \mu, \Pi \rangle$ where $\theta = \sigma'\nu$, $\sigma = \theta \upharpoonright \text{var}(G)$ and $\mu = \theta \iota^{-1} \upharpoonright \text{var}(G)$. Note that μ is well-defined thanks to Lemma 4.3. \square

The next theorem ensures that SQCLP-AGSS is correct provided that CLP-AGSS is also correct. The proof relies on the semantic results of the two previous subsections.

Theorem 4.5

Assume that CLP-AGSS is correct (in the sense of Definition 3.3). Let SQCLP-AGSS be as in the previous definition. Then SQCLP-AGSS is correct in the sense of Definition 2.3.

Proof

We separately prove that SQCLP-AGSS is *sound* and *weakly complete*.

— *SQCLP-AGSS is sound.* Let $\langle \sigma, \mu, \Pi \rangle$ be an answer computed by SQCLP-AGSS for G, \mathcal{P} . We must prove that $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$. By Definition 4.4 we can assume $\langle \sigma', \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ and $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ such that $\sigma = \theta \upharpoonright \text{var}(G)$ and $\mu = \theta \iota^{-1} \upharpoonright \text{var}(G)$ with $\theta = \sigma'\nu$. Because of Lemma 4.3 we have $\langle \theta, \Pi \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ and $W\theta \in \text{ran}(\iota)$ for every $W \in \text{var}(G'') \setminus \text{var}(G)$. Note that:

- $\theta =_{\text{var}(G')} \sigma$. This follows from $\text{var}(G') = \text{var}(G)$ and the construction of σ .
- $\theta =_{\text{var}(G')} \mu \iota$. This follows from $\text{var}(G') = \text{var}(G)$ and $\theta =_{\text{var}(G)} \mu \iota$, that is obvious from the construction of μ .
- $W\theta \in \text{ran}(\iota)$ for each $W \in \text{var}(G'') \setminus (\text{var}(G') \cup \text{var}(G'))$. This is a consequence of Lemma 4.3 since $\text{var}(G'') \setminus (\text{var}(G') \cup \text{var}(G')) \subseteq \text{var}(G'') \setminus \text{var}(G')$ and $\text{var}(G') = \text{var}(G)$.

From the previous items and Theorem 4.4 we get $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}'}(G')$, which trivially implies $\langle \sigma, \mu, \Pi \rangle \in \text{Sol}_{\mathcal{P}}(G)$ because of Theorem 4.2.

— *SQCLP-AGSS is weakly complete.* Let $\langle \eta, \rho, \emptyset \rangle \in \text{GSol}_{\mathcal{P}}(G)$ be a ground solution for G w.r.t. \mathcal{P} . We must prove that it is subsumed—in the sense of Definition 2.2(3)—by some answer $\langle \sigma, \mu, \Pi \rangle$ computed by SQCLP-AGSS for G, \mathcal{P} .

By Theorem 4.2 we have that $\langle \eta, \rho, \emptyset \rangle$ is also a ground solution for G' w.r.t. \mathcal{P}' . In addition, by Theorem 4.4 $\langle \eta', \emptyset \rangle \in \text{Sol}_{\mathcal{P}''}(G'')$ for some η' such that

- (1) $\eta' =_{\text{var}(G')} \eta$,
- (2) $\eta' =_{\text{var}(G')} \rho$ and hence $\eta'(\iota^{-1}) =_{\text{var}(G')} \rho$, and
- $W\eta' \in \text{ran}(\iota)$ for each $W \in \text{var}(G'') \setminus (\text{var}(G') \cup \text{var}(G'))$ (i.e. $w'_i \eta' \in \text{ran}(\iota)$ for each $i = 1 \dots m$ such that w'_i is a variable).

By construction of η' , it is clear that $\langle \eta', \emptyset \rangle$ is ground. Now, by the weak completeness of CLP-AGSS, there is some computed answer $\langle \sigma', \Pi \rangle$ subsuming $\langle \eta', \emptyset \rangle$, therefore satisfying

- (3) there is some $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$, and
- (4) $\eta' =_{\text{var}(G'')} \sigma' \nu$.

Because of Definition 4.4 one can build a SQCLP-AGSS computed answer $\langle \sigma, \mu, \Pi \rangle$ as follows:

- (5) $\sigma = \sigma' \nu \upharpoonright \text{var}(G)$
- (6) $\mu = \sigma' \nu \iota^{-1} \upharpoonright \text{var}(G)$

We now check that $\langle \sigma, \mu, \Pi \rangle$ subsumes $\langle \eta, \rho, \emptyset \rangle$:

- $W_i \rho \leq W_i \mu$ and even $W_i \rho = W_i \mu$ because:

$$W_i \rho =_{(2)} W_i \eta'(\iota^{-1}) =_{(4)} W_i \sigma' \nu(\iota^{-1}) =_{(6)} W_i \mu .$$

- $\nu \in \text{Sol}_{\mathcal{C}}(\Pi)$ by (3) and, moreover, for any $X \in \text{var}(G)$:

$$X\eta =_{(1)} X\eta' =_{(4)} X\sigma' \nu =_{(\dagger)} X\sigma' \nu \nu =_{(5)} X\sigma \nu$$

therefore $\eta =_{\text{var}(G)} \sigma \nu$.

The step (\dagger) is justified because $\nu \in \text{Val}_{\mathcal{C}}$ implies $\nu = \nu \nu$. \square

5 A Practical Implementation

This section is devoted to the more practical aspects of the SQCLP programming scheme and it is developed in three subsections: Subsection 5.1 explains what steps must be given when implementing a programming scheme like this and why the theoretic results presented in the previous sections—with special emphasis in those in Subsection 4.3—become useful for implementation. Subsection 5.2 introduces a prototype implementation and explains how to write programs and how to solve goals. Finally, in Subsection 5.3 we study the unavoidable overload introduced in the system by qualifications and proximity relations when comparing the execution of programs without any explicit use of such resources.

5.1 SQCLP over a CLP Prolog System

Assume an available CLP Prolog System, a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} and a goal G for \mathcal{P} . Our purpose is to implement a goal solving system for SQCLP following Definition 4.4. We will examine each step in this schema, discussing the necessary implementation details for putting theory into practice.

The first step is to obtain the transformed programs $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ and $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$; and the transformed goals $G' = \text{elim}_{\mathcal{S}}(G)$ and $G'' = \text{elim}_{\mathcal{D}}(G')$. According to Definition 4.2(3), $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ is of the form $EQ_{\mathcal{S}} \cup \hat{\mathcal{P}}_{\mathcal{S}}$, where $EQ_{\mathcal{S}}$ is obtained following Definition 4.1 and $\hat{\mathcal{P}}_{\mathcal{S}}$ is obtained following Definition 4.2(3,2). When implementing $EQ_{\mathcal{S}}$ a first difficulty arises, namely the implementation of $\sim \in DP^2$, which apparently requires one clause of the form:

$$u \sim u' \stackrel{\mathbf{t}}{\leftarrow} \text{pay}_{\lambda} \#?$$

for each pair $u, u' \in B_{\mathcal{C}}$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$, and one clause of the form:

$$c(\bar{X}_n) \sim c'(\bar{Y}_n) \stackrel{\mathbf{t}}{\leftarrow} \text{pay}_{\lambda} \#?, ((X_i \sim Y_i) \#?)_{i=1 \dots n}$$

for each pair $c, c' \in DC^n$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{b}$. While this should obviously require an infinite number of clauses (because DC^n is infinite and $\mathcal{S}(c, c) = \mathbf{t} \neq \mathbf{b}$ for all $c \in DC^n$; and also $B_{\mathcal{C}}$ is infinite—in general—and $\mathcal{S}(u, u) = \mathbf{t} \neq \mathbf{b}$ for every $u \in B_{\mathcal{C}}$), in practice, it is enough to limit the number of clauses to the finite number of different basic values $u \in B_{\mathcal{C}}$ and constructors $c \in DC^n$ that can be found either in \mathcal{P} , G or \mathcal{S} .

A similar difficulty arises when codifying the clauses for predicates $\text{pay}_{\lambda} \in DP^0$, which according to Definition 4.1 there should be a clause of the form:

$$\text{pay}_{\lambda} \stackrel{\lambda}{\leftarrow}$$

in $EQ_{\mathcal{S}}$ for each $\lambda \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$. In this case, the solution is also similar because it suffices to generate enough pay_{λ} clauses for the finite $\lambda \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ that can be found occurring either in the clauses of $\hat{\mathcal{P}}_{\mathcal{S}}$ or in the clauses implementing the predicate $\sim \in DP^2$.

The construction of $\hat{\mathcal{P}}_{\mathcal{S}}$, following Definition 4.2, presents no particular difficulties. For each clause $C : (p(\bar{t}_n) \stackrel{\alpha}{\leftarrow} \bar{B}) \in \mathcal{P}$ we will generate a finite set $\hat{\mathcal{C}}_{\mathcal{S}}$ of clauses, because the number of symbols p' such that $\mathcal{S}(p, p') = \lambda \neq \mathbf{b}$ will be also finite in practice. Finally, the construction of G' is merely the straightforward replacement of all the occurrences of ‘==’ in G by ‘ \sim ’.

The transformation $\text{elim}_{\mathcal{D}}$ from QCLP(\mathcal{D}, \mathcal{C}) into CLP(\mathcal{C}), is defined in Definition 4.3. $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$ is obtained by incorporating the two clauses of the program $E_{\mathcal{D}}$ to the result of applying the transformation rules in Figure 5 to the QCLP(\mathcal{D}, \mathcal{C})-program \mathcal{P}' . Applying the transformation rules is straightforward, but the codification of constraints $\text{qVal}(X)$ and $\text{qBound}(X, Y, Z)$ in $E_{\mathcal{D}}$ requires some clarification. In our implementation we have considered the constraint domain \mathcal{R} , as well as any qualification domain that can be built from \mathcal{B}, \mathcal{U} and \mathcal{W} by means of the strict cartesian product operation \otimes including, in particular, $\mathcal{U} \otimes \mathcal{W}$. These qualification domains are existentially expressible in \mathcal{R} , therefore the constraints

can be implemented by defined predicates as explained in Section 4.2. In particular in our prototype implementation these predicates are:

```
% qval( +QDom, ?W ):
qval(b, 1).
qval(u, W) :- {W > 0, W =< 1}.
qval(w, W) :- {W >= 0}.
qval((D1,D2), (W1,W2)) :- qval(D1, W1), qval(D2, W2).

% qbound( +QDom, ?X, ?Y, ?Z ):
qbound(b, 1, 1, 1).
qbound(u, X, Y, Z) :- {X =< Y * Z}.
qbound(w, X, Y, Z) :- {X >= Y + Z}.
qbound((D1,D2), (X1,X2), (Y1,Y2), (Z1,Z2)) :- qbound(D1, X1, Y1, Z1),
qbound(D2, X2, Y2, Z2).
```

Instead of using different $qVal$ and $qBound$ predicates for each allowable \mathcal{D} , our prototype implementation just uses two predicates $qVal$ and $qBound$ with an extra first argument, used to encode an identifier of some specific allowable \mathcal{D} . This parameter can take either the value **b** (for \mathcal{B}), **u** (for \mathcal{U}), **w** (for \mathcal{W}) or a pair (D_1, D_2) (for $\mathcal{D}_1 \otimes \mathcal{D}_2$), where each D_i can be either **b**, **u**, **w** or another pair representing a product. For instance $((u, w), w)$ represents the qualification domain $(\mathcal{U} \otimes \mathcal{W}) \otimes \mathcal{W}$. The compiler ensures that this argument takes the correct value for each transformed program and goal depending on the specific instance of the SQCLP scheme the program is written for.

After obtaining \mathcal{P}'' and G'' , the CLP Prolog System is used to solve G'' w.r.t. \mathcal{P}'' . This yields computed answers of the form $\langle \sigma', \Pi \rangle$. Now, instead of obtaining particular substitutions $\theta = \sigma' \nu$, $\sigma = \theta \upharpoonright \text{var}(G)$ and $\mu = \theta_i^{-1} \upharpoonright \text{war}(G)$ for any $\nu \in \text{Sol}_{\mathcal{L}}(\Pi)$ as explained in Definition 4.4(2), our prototype implementation limits itself to display $\langle \sigma', \Pi \rangle$ as the computed answer in SQCLP. The reason behind this behavior is that, in general (and particularly in \mathcal{R}), it is impossible to enumerate the possible solutions $\nu \in \text{Sol}_{\mathcal{L}}(\Pi)$. Thus, it results impossible to implement a technique for obtaining all the possible triples $\langle \sigma, \mu, \Pi \rangle$. Note, however, that for a user it will not be difficult to distinguish, in the shown computed answers, what variable bindings correspond to the substitution σ of the triple and what to the substitution μ , even when the qualification variables are not bound but constrained, which is a common behavior in the context of CLP programming.

However, for the SQCLP-AGSS of Definition 4.4, it results mandatory to define the computed answers in terms of $\nu \in \text{Sol}_{\mathcal{L}}(\Pi)$, because our SQCLP-semantics relies on proving instances of G for some specific ground values of the variables in $\text{war}(G)$.

5.2 (S)QCLP: A Prototype System for SQCLP Programming

The prototype implementation object of this subsection is publicly available, and can be found at:

<http://gpd.sip.ucm.es/cromdia/qclp>

The system currently requires the user to have installed either *SICStus Prolog* or *SWI-Prolog*, and it has been tested to work under Windows, Linux and MacOSX platforms. The latest version available at the time of writing this paper is 0.6. If a latter version is available some things might have changed but in any case the main aspects of the system should remain the same. Please consult the *changelog* provided within the system itself for specific changes between versions.

SQCLP is a very general programming scheme and, as such, it supports different proximity relations, different qualification domains and different constraint domains when building specific instances of the scheme for any specific purpose. As it would result impossible to provide an implementation for every admissible triple (or instance of the scheme), it becomes mandatory to decide in advance what specific instances will be available for writing programs in (S)QCLP. In essence:

1. In its current state, the only available constraint domain is \mathcal{R} . Thus, under both *SICStus Prolog* and *SWI-Prolog* the library `clpr` will provide all the available primitives in (S)QCLP programs.
2. The available qualification domains are: ‘b’ for the domain \mathcal{B} ; ‘u’ for the domain \mathcal{U} ; ‘w’ for the domain \mathcal{W} ; and any strict cartesian product of those, as e.g. ‘(u,w)’ for the product domain $\mathcal{U} \otimes \mathcal{W}$.
3. With respect to proximity relations, the user will have to provide, in addition to the two symbols and their proximity value, their *kind* (either predicate or constructor) and their *arity*. Both kind and arity must be the same for each pair of symbols having a proximity value different of b.

Note, however, that when no specific proximity relation \mathcal{S} is provided for a given program, \mathcal{S}_{id} is then assumed. Under this circumstances, an obvious technical optimization consists on transforming the original program only with $\text{elim}_{\mathcal{D}}$, thus reducing the overload introduced in this case by $\text{elim}_{\mathcal{S}}$. The reason behind this optimization is that for any given SQCLP($\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C}$)-program \mathcal{P} , it is also true that \mathcal{P} is a QCLP(\mathcal{D}, \mathcal{C})-program, therefore $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$ must semantically be equivalent to $\text{elim}_{\mathcal{D}}(\mathcal{P})$. Nevertheless, $\text{elim}_{\mathcal{D}}(\mathcal{P})$ behaves more efficiently than $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$ due to the reduced number of resulting clauses. Thus, in order to improve the efficiency, the system will avoid the use of $\text{elim}_{\mathcal{S}}$ when no proximity relation is provided by the user.

The final available instances in the (S)QCLP system are: SQCLP($\mathcal{S}, \text{b}, \text{clpr}$), SQCLP($\mathcal{S}, \text{u}, \text{clpr}$), SQCLP($\mathcal{S}, \text{w}, \text{clpr}$), SQCLP($\mathcal{S}, (\text{u}, \text{w}), \text{clpr}$), ... and their counterparts in the QCLP scheme when $\mathcal{S} = \mathcal{S}_{\text{id}}$.

5.2.1 Programming in (S)QCLP

Programming in (S)QCLP is straightforward if the user is accustomed to the Prolog programming style. However, there are three syntactic differences with pure Prolog:

1. Clauses implications are replaced by “ $\leftarrow d$ ” where $d \in D \setminus \{\text{b}\}$. If $d = \text{t}$, then the implication can become just “ \leftarrow ”. E.g. “ $\leftarrow 0.9$ ” is a valid implication in the domains \mathcal{U} and \mathcal{W} ; and “ $\leftarrow (0.9, 2)$ ” is a valid implication in the domain $\mathcal{U} \otimes \mathcal{W}$.

2. Clauses in (S)QCLP are not finished with a dot (.). They are separated by layout, therefore all clauses in a (S)QCLP program must start in the same column. Otherwise, the user will have to explicitly separate them by means of semicolons (;).
3. After every body atom (even constraints) the user can provide a threshold condition using '#'. The notation '?' can also be used instead of some particular qualification value, but in this case the threshold condition '#?' can be omitted.

Comments are as in Prolog:

```
% This is a line comment.
/* This is a multi-line comment, /* and they nest! */. */
```

and the basic structure of a (S)QCLP program is the following (line numbers are for reference):

File: *Peano.qclp*

```
1  % Directives...
2  # qdom w

3  % Program clauses...
4  % num( ?Num )
5  num(z) <--
6  num(s(X)) <-1- num(X)
```

In the previous small program, lines 1, 3 and 4 are line comments, line 2 is a program directive telling the compiler the specific qualification domain the program is written for, and lines 5 and 6 are program clauses defining the well-known Peano numbers. As usual, comments can be written anywhere in the program as they will be completely ignored (remember that a line comment must necessarily end in a new line character, therefore the very last line of a file cannot contain a line comment), and directives must be declared before any program clause. There are three program directives in (S)QCLP:

1. The first one is “#qdom *qdom*” where *qdom* is any system available qualification domain, i.e. *b*, *u*, *w*, (*u,w*)... See line 2 in the previous program sample as an example. This directive is mandatory because the user must tell the compiler for which particular qualification domain the program is written.
2. The second one is “#prox *file*” where *file* is the name of a file (with extension *.prox* containing a proximity relation. If the name of the file starts with a capital letter, or it contains spaces or any special character, *file* will have to be quoted with single quotes. For example, assume that with our program file we have another file called *Proximity.prox*. Then, we would have to write “#prox ‘Proximity’” to link the program with such proximity relation. This directive is optional, and if omitted, the system assumes that the program is of an instance of the QCLP scheme.
3. The third one is “#optimized_unif”. This directive tells the compiler that the program is intended to be used with the optimized version of the unification algorithm, what improves the general efficiency of the goal solving

process. However, as noted at the end of Section 2, this could have the effect of losing valid answers, although we conjecture that if the proximity relation is transitive and if the program clauses do not make use of attenuation factors other than \mathbf{t} , this will not happen.

Proximity relations are defined in files of extension `.prox` with the following form:

File: *Work.prox*

```

1 % Predicates: pprox( S1, S2, Arity, Value ).
2 pprox(wrote, authored, 2, (0.9,0)).

3 % Constructors: cprox( S1, S2, Arity, Value ).
4 cprox(king_lear, king_liar, 0, (0.8,2)).
```

where the file can contain `pprox/4` Prolog facts, for defining proximity between predicate symbols of any arity; or `cprox/4` Prolog facts, for defining proximity between constructor symbols of any arity. The arguments of both `pprox/4` and `cprox/4` are: the two symbols, their arity and its proximity value. Note that, although it is not made explicit the qualification domain this proximity relation is written for, all values in it must be of the same specific qualification domain, and this qualification domain must be the same declared in every program using the proximity relation. Otherwise, the solving of equations may produce unexpected results or even fail.

Reflexive and symmetric closure is inferred by the system, therefore, there is no need for writing reflexive proximity facts, nor the symmetric variants of proximity facts already provided. You can notice this in the previous sample file in which neither reflexive proximity facts, nor the symmetric proximity facts to those at lines 2 and 4 are provided. In the case of being explicitly provided, additional (repeated) solutions might be computed for the same given goal, although soundness and weak completeness of the system should still be preserved. Transitivity is neither checked nor inferred so the user will be responsible for ensuring it if desired.

As the reader would have already guessed, the file `Work.prox` implements the proximity relation \mathcal{S}_r of Example 4.1 in (S)QCLP. Finally, the program \mathcal{P}_r of Example 4.1 can be represented in (S)QCLP as follows:

File: *Work.qclp*

```

1 # qdom (u,w)
2 # prox 'Work'

3 % famous( ?Author )
4 famous(shakespeare) <-(0.9,1)-

5 % wrote( ?Author, ?Book )
6 wrote(shakespeare, king_lear) <-(1,1)-
7 wrote(shakespeare, hamlet) <-(1,1)-

8 % good_work( ?Work )
9 good_work(X) <-(0.75,3)- famous(Y)#(0.5,100), authored(Y,X)
```

Note that, at line 1 the qualification domain $\mathcal{U} \otimes \mathcal{W}$ is declared, and at line 2 the proximity relation at `Work.prox` is linked to the program. In addition, observe

that one threshold constraint is imposed for a body atom in the program clause at line 9, effectively requiring to prove `famous(Y)` for a qualification value of *at least* $(0.5, 100)$ to be able to use this program clause.

Finally, we explain how constraints are written in (S)QCLP. As it has already been said, only \mathcal{R} is available, thus both in *SICStus Prolog* and *SWI-Prolog* the library `clpr` is the responsible for providing the available primitive predicates. Given that constraints are primitive atoms of the form $\mathbf{r}(\bar{\mathbf{t}}_n)$ where $\mathbf{r} \in PP^n$ and \mathbf{t}_i are terms; primitive atoms share syntax with usual Prolog atoms. At this point, and having that many of the primitive predicates are syntactically operators (hence not valid identifiers), the syntax for predicate symbols has been extended to include operators, therefore predicate symbols like $op_+ \in PP^3$, which codifies the operation $+$ in a 3-ary predicate, will let us to build constraints of the form $+(A, B, C)$, that must be understood as in $A + B = C$ or $C = A + B$. Similarly, predicate symbols like $cp_> \in PP^2$, which codifies the comparison operator $>$ in a binary predicate, will let us to build constraints of the form $>(A, B)$, that must be understood as in $A > B$. Any other primitive predicate such as $maximize \in PP^1$, will let us to build constraints like $maximize(X)$. Valid primitive predicate symbols include $+$, $-$, $*$, $/$, $>$, $>=$, $=$, $<$, $maximize$, $minimize$, etc.

Threshold constraints can also be provided for primitive atoms in the body of clauses with the usual notation. Note, however, that due the semantics of SQCLP, all primitive atoms can be trivially proved with \mathbf{t} if they ever succeeds—so threshold constraints become, in this case, of no use.

The syntax for constraints explained above follows the standard syntax for atoms. Nonetheless, the system also allows to write these constraints in a more natural infix notation. More precisely, $+(A, B, C)$ can be also written in the infix form $A+B=C$ or $C=A+B$, and $>(X, Y)$ in the infix form $X>Y$; and similarly for other op and cp constraints. When using infix notation, threshold conditions can be set by (optionally) enclosing the primitive atom between parentheses, therefore becoming $(A+B=C)\#t$, $(C=A+B)\#t$ or $(X>Y)\#t$ (or any other valid qualification value or $?$). Using parentheses is recommended to avoid understanding that the threshold condition is set only for the last term in the constraint, which would not be the case. Note that even in infix notation, operators cannot be nested, that is, terms A , B , C , X and Y cannot have operators as main symbols (neither in prefix nor in infix notation), so the infix notation is just a syntactic sugar of its corresponding prefix notation.

As a final example for constraints, one could write the predicate `double/2` in (S)QCLP, for computing the double of any given number, with just the clause `double(N,D) <-- *(N,2,D)`, or `double(N,D) <-- N*2=D` for a clause with a more natural syntax.

5.2.2 The interpreter for (S)QCLP

The interpreter for (S)QCLP has been implemented on top of both *SICStus Prolog* and *SWI-Prolog*. To load it, one must first load her desired (and supported) Prolog system and then load the main file of the interpreter—i.e. `qclp.pl`—, that will be located in the main (S)QCLP folder among other folders. Once loaded, one will

see the welcome message and will be ready to compile and load programs, and to execute goals.

WELCOME TO (S)QCLP 0.6

(S)QCLP is free software and comes with absolutely no warranty.

Support & Updates: <http://gpd.sip.ucm.es/cromdia/qclp>.

Type `':help.'` for help.

yes

| ?-

From the interpreter for (S)QCLP one can, in addition to making use of any standard Prolog goals, use the specific (S)QCLP commands required for both interacting with the (S)QCLP system, and for compiling/loading SQCLP programs. All these commands take the form:

`:command.`

if they do not require arguments, or:

`:command(Arg1, ..., Argn).`

if they do; where each argument *Arg_i* must be a prolog atom unless stated otherwise. The most useful commands are:

- `:cd(Folder).`
Changes the working directory to *Folder*. *Folder* can be an absolute or relative path.
- `:compile(Program).`
Compiles the (S)QCLP program '*Program.qclp*' producing the equivalent Prolog program in the file '*Program.pl*'.
- `:load(Program).`
Loads the already compiled (S)QCLP program '*Program.qclp*' (note that the file '*Program.pl*' must exist for the program to correctly load).
- `:run(Program).`
Compiles the (S)QCLP program '*Program.qclp*' and loads it afterwards. This command is equivalent to executing: `:compile(Program), :load(Program).`

For illustration purposes, we will assume that you have the files `Work.prox` and `Work.qclp` (both as seen before) in the folder `~/examples`. Under these circumstances, after loading your preferred Prolog system and the interpreter for (S)QCLP, one would only have to change the working directory to that where the files are located:

| ?- `:cd('~/examples').`

and run the program:

| ?- `:run('Work').`

If no errors are encountered, one should see the output:

```
| ?- :run('Work').
<Work> Compiling...
<Work> QDom: 'u,w'.
<Work> Prox: 'Work'.
<Work> Translating to QCLP...
<Work> Translating to CLP...
<Work> Generating code...
<Work> Done.
<Work> Loaded.
yes
```

and now everything is ready to execute goals for the program loaded.

5.2.3 Executing SQCLP-Goals

Recall that goals have the form $A_1\#W_1, \dots, A_m\#W_m \parallel W_1 \triangleright^? \beta_1, \dots, W_m \triangleright^? \beta_m$ which in actual (S)QCLP syntax becomes:

```
| ?- A1#W1, ..., Am#Wm :: W1 >= B1, ..., Wm >= Bm.
```

Note the following:

1. Goals must end in a dot (.).
2. The symbol ' \parallel ' is replaced by ' $::$ '.
3. The symbol ' $\triangleright^?$ ' is replaced by ' $>=$ ' (and this is independent of the qualification domain in use, so that it may mean \leq in \mathcal{W}).
4. Conditions of the form $W \triangleright^? ?$ *must be omitted*, therefore $A_1\#W_1, A_2\#W_2 \parallel W_1 \triangleright^? ?, W_2 \triangleright^? \beta_2$ becomes " $A1\#W1, A2\#W2 :: W2 >= B2.$ ", and $A\#W \parallel W \triangleright^? ?$ becomes just " $A\#W.$ ".

Assuming now that we have loaded the program `Work.qclp` as explained before, we can execute the goal $good_work(king_liar)\#W \parallel W \triangleright^? (0.5, 100)$:

```
| ?- good_work(king_liar)#W::W>=(0.5,10).
W = (0.6,5.0) ?
yes
```

5.2.4 Examples

To finish this subsection, we are now showing some additional goal executions using the interpreter for (S)QCLP and the programs displayed along the paper.

Peano. Consider the program `Peano.qclp` as displayed at the beginning of Subsection 5.2.1. Qualifications in this program are intended as a cost measure for obtaining a given number in the Peano representation, assuming that each use of the clause at line 6 requires to pay *at least* 1. In essence, threshold conditions will impose an upper bound over the maximum number obtainable in goals containing the atom `num(X)`. Therefore if we ask for numbers *up to* a cost of 3 we get the following answers:

```
Goal    ?- num(X)#W: :W>=3.

Sol1    W = 0.0, X = z ? ;
Sol2    W = 1.0, X = s(z) ? ;
Sol3    W = 2.0, X = s(s(z)) ? ;
Sol4    W = 3.0, X = s(s(s(z))) ? ;
no
```

Work. Consider now the program `Work.qclp` and the proximity relation `Work.prox`, both as displayed in Subsection 5.2.1 above. In this program, qualifications behave as the conjunction of the certainty degree of the user confidence about some particular atom, and a measure of the minimum cost to pay for proving such atom. In these circumstances, we could ask—just for illustration purposes—for famous authors with a minimum certainty degree—for them being actually famous—of 0.5, and with a proof cost of no more than 30 (think of an upper bound for possible searches in different databases). Such a goal would have, in this very limited example, only the following solution:

```
Goal    ?- famous(X)#W: :W>=(0.5,30).

Sol1    W = (0.9,1.0), X = shakespear ? ;
no
```

meaning that we can have a confidence of `shakespear` being famous of 0.9, and that we can prove it with a cost of 1.

Now, in a similar fashion we could try to obtain different works that can be considered as good works by using the last clause in the example. Limiting the search to those works that can be considered good with a qualification value better or equal to (0.5,100) produce the following result:

```
Goal    ?- good_work(X)#W: :W>=(0.5,100).

Sol1    W = (0.675,4.0), X = king_lear ? ;
Sol2    W = (0.6,5.0), X = king_liar ? ;
no
```

It is important to remark here that the qualification value obtained for a particular computed answer is not guaranteed to be the best possible one; rather, different computed answers may compute different qualification values which can be observed by the user. This is easy to see if we try to solve a more particular goal:

```
Goal    ?- good_work(king_liar)#W: :W>=(0.675,4.0).

Sol1    W = (0.675,4.0) ? ;
no
```

That is, not only `good_work(king_liar)` can be proved for `W = (0.6,5.0)` as shown in `Sol2` above, but also with `W = (0.675,4.0)`, which results a better qualification value (i.e. greater certainty degree and lower proof cost).

Library. Finally, consider the program \mathcal{P}_s and the proximity relation \mathcal{S}_s , both as displayed in Figure 1 of Section 2. As it has been said when this example was introduced, the predicate `guessRdrLvl` takes advantage of attenuation factors to

encode heuristic rules to compute reader levels on the basis of vocabulary level and other book features. As an illustration of use, consider the following goal:

```
Goal    ?- guessRdrLvl(book(2, 'Dune', 'F. P. Herbert', english, sciFi,
                        medium, 345), Level)#W.

Sol1    W = 0.8, Level = intermediate ? ;
        ...
Sol6    W = 0.7, Level = upper ?
        yes
```

Here we ask for possible ways of classifying the second book in the library according to reader levels. We obtain as valid solutions, among others, *intermediate* with a certainty factor of 0.8; and *upper* with a certainty factor of 0.7. These valid solutions show that the predicate *guessRdrLvl* tries with different levels for any certain book based on the heuristic implemented by the qualified clauses.

To conclude, consider now the goal proposed in Section 2 for this program. For such goal we obtain:

```
Goal    ?- search(german, essay, intermediate, ID)#W::W>=0.65.

Sol1    W = 0.8, ID = 4 ?
        yes
```

What tells us that the forth book in the library is written in German, it can be considered to be an essay, and it is targeted for an intermediate reader level. All this with a certainty degree of *at least* 0.8.

5.3 Efficiency

The minimum—and unavoidable—overload introduced by qualifications and proximity relations in the transformed programs manifests itself in the case of (S)QCLP programs which use the identity proximity relation and have *t* as the attenuation factor of all their clauses. In order to measure this overload we have made some experiments using some program samples, taken from the *SICStus Prolog Benchmark* that can be found in:

<http://www.sics.se/isl/sicstuswww/site/performance.html>

and we have compared the time it took to repeatedly execute a significant number of times each program in both (S)QCLP and *SICStus Prolog* making use of a *slightly* modified (to ensure a correct behavior in both systems) version of the harness also provided in the same site.

From all the programs available in the aforementioned site, we selected the following four:

- *naivrev*: naive implementation of the predicate that reverses the contents of a list.
- *deriv*: program for symbolic derivation.
- *qsort*: implementation of the well-known sorting algorithm *Quicksort*.
- *query*: obtaining the population density of different countries.

No other program could be used because they included impure features such as cuts which are not currently supported by our system. In order to adapt these Prolog programs to our setting the following modifications were required:

1. All the program clause are assumed to have `t` as attenuation factor. After including these attenuation factors, we obtain as results QCLP programs. More specifically we obtain two QCLP programs for each initial Prolog program, one using the qualification domain \mathcal{B} (because this domain uses trivial constraints), and another using the qualification domain \mathcal{U} (which uses \mathcal{R} -constraints).
2. We define an empty proximity relation, allowing us to obtain two additional SQCLP-programs.
3. By means of the program directive “`#optimized.unif`” defined in Subsection 5.2.1, each SQCLP program can be also executed in this optimized mode. Therefore each original Prolog Program produces six (S)QCLP programs, denoted as Q(b), Q(u), PQ(b), PQ(u), SQ(b) and SQ(u) in Table 1.

Additionally some minor modifications to the program samples have been introduced for compatibility reasons, i.e. additions using the predicate `is/2` were replaced, both in the Prolog version of the benchmark and in the multiple (S)QCLP versions, by `clpr` constraints. In any case, all the program samples used for this benchmarks in this subsection can be found in the folder `benchmarks/` of the (S)QCLP distribution.

Finally, we proceeded to solve the same goals for every version of the benchmark programs, both in *SICStus Prolog* and in (S)QCLP. The benchmark results can be found in Table 1. All the experiments were performed in a computer with a Intel(R) Core(TM)2 Duo CPU at 2.19GHz and with 3.5 GB RAM.

Table 1. Time overload factor with respect to Prolog

Program	Q(b) ^a	Q(u) ^b	PQ(b) ^c	PQ(u) ^d	SQ(b) ^e	SQ(u) ^f
naivrev	1.80	10.71	4289.79	4415.11	56.22	65.75
deriv	1.94	10.60	331.45	469.67	29.63	39.32
qsort	1.05	1.11	135.59	136.98	2.51	2.83
query	1.02	1.12	7.17	7.13	3.80	3.88

^a QCLP(\mathcal{B}, \mathcal{R}) version (i.e. the program does not have the `#prox` directive).

^b QCLP(\mathcal{U}, \mathcal{R}) version (i.e. the program does not have the `#prox` directive).

^c SQCLP($\mathcal{S}_{id}, \mathcal{B}, \mathcal{R}$) version.

^d SQCLP($\mathcal{S}_{id}, \mathcal{U}, \mathcal{R}$) version.

^e SQCLP($\mathcal{S}_{id}, \mathcal{B}, \mathcal{R}$) version with directive `#optimized.unif`.

^f SQCLP($\mathcal{S}_{id}, \mathcal{U}, \mathcal{R}$) version with directive `#optimized.unif`.

The results in the table indicate the slowdown factor obtained for each version of each program. For instance, the first column indicates that the time required for evaluating the goal corresponding to the sample program *naivrev* in QCLP(\mathcal{B}, \mathcal{R}) is about 1.80 times the required time for the evaluation of the same goal in Prolog. Next we discuss the results:

- *Influence of the qualification domain.* In general the difference between the slowdown factors obtained for the two considered qualification domains is not large. However, in the case of QCLP-programs *naivrev* and *deriv* the difference increases notably. This is due to the different ratios of the \mathcal{B} -constraints w.r.t. the program and \mathcal{U} -constraints w.r.t. the program. It must be noticed that the transformed programs are the same in both cases, but for the implementation of `qval` and `qbound` constraints, which is more complex for \mathcal{U} as one can see in Section 5.1. In the case of *naivrev* and *deriv* this makes a big difference because the number of computation steps directly required by the programs is much smaller than in the other cases. Thus the slowdown factor becomes noticeable for the qualification domain \mathcal{U} in computations that requires a large number of steps.
- *Influence of the proximity relation.* The introduction of a proximity relation, even of empty, is very significative. This is due to the introduction of the predicate \sim , which replaces Prolog unification. The situation even worsens when the computation introduces large constructor terms, as in the case of *naivrev* which deals with Prolog lists. The efficient Prolog unification is replaced by an explicit term decomposition.
- *Influence of the optimized unification.* As explained at the end of Section 2 this optimization can lead to the loss of solutions in general. However, this is not the case for the chosen examples. As seen in the table, the use of the program directive `#optimized_unif` causes a clear increase in the efficiency of goal solving for these examples.

6 Conclusions

In our recent work (Rodríguez-Artalejo and Romero-Díaz 2010a) we extended the classical CLP scheme to a new programming scheme SQCLP whose instances $\text{SQCLP}(\mathcal{S}, \mathcal{D}, \mathcal{C})$ were parameterized by a proximity relation \mathcal{S} , a qualification domain \mathcal{D} and a constraint domain \mathcal{C} . This new scheme offered extra facilities for dealing with expert knowledge representation and flexible query answering. In this paper we have contributed to the aforementioned scheme providing, in a more practical sense, both a semantically correct transformation technique, in two steps, for transforming SQCLP programs and goals into equivalent CLP programs and goals; and a prototype implementation on top CLP(\mathcal{R}) systems like *SICStus Prolog* and *SWI-Prolog* of some particularly interesting instances of the scheme.

The two-step transformation technique presented in Section 4 has provided us with the needed theoretical results for effectively showing how proximity relations can be reduced to qualifications and clause annotations by means of the transformation $\text{elim}_{\mathcal{S}}$; and how qualifications and clause annotations can be reduced to classical CLP programming by means of the transformation $\text{elim}_{\mathcal{D}}$. These two transformations altogether, ultimately enables the use of the classical mechanism of SLD resolution to obtain computed answers for SQCLP goals w.r.t SQCLP programs, via their equivalent CLP programs and goals and the computed answers obtained from them by any capable CLP goal solving procedure.

The prototype implementation presented in Section 5 has finally allowed us to execute all the examples showed in this paper—and in previous ones—, and a series of benchmarks for measuring the overload actually introduced by proximity relations—or by similarity relations—and by clause annotations and qualifications. While we are aware that the prototype implementation presented in this paper has to be considered a research application (and as such, we have to admit that it cannot be used for industrial applications), we think that it can contribute to the field as a quite complete implementation of an extension of the CLP(\mathcal{R}) scheme with proximity relations and qualifications. Some related implementation techniques and systems have been cited in the introduction. However, as far as we know, no other implementation in this field has ever provided support for proximity (and similarity) relations, qualifications via clause annotations and CLP(\mathcal{R}) style programming. Moreover, our results in Section 4 on the semantic correctness of our implementation technique are in our opinion another contribution of this paper which has no counterpart in related approaches.

In the future, and taking advantage of the prototype system we have already developed, we plan to investigate possible applications which can profit from proximity relations and qualifications, such as in the area of flexible query answering. In particular, we plan to investigate application related to flexible answering of queries to XML documents, in the line of (Campi et al. 2009) and other related papers. As support for practical applications, we also plan to increase the repertoire of constraint and qualification domains which can be used in the (S)QCLP prototype, adding the constraint domain \mathcal{FD} and the qualification domain \mathcal{W}_d defined in Section 2.2.3 of (Rodríguez-Artalejo and Romero-Díaz 2010b). On a more theoretical line, other possible lines of future work include: a) extension of the SLD(\mathcal{D}) resolution procedure presented in (Rodríguez-Artalejo and Romero-Díaz 2008) to a SQCLP goal solving procedure able to work with constraints and a proximity relation; b) investigation of the conjecture stated at the end of Section 2; and c) extension of the QCFLP (*qualified constraint functional logic programming*) scheme in (Caballero et al. 2009) to work with a proximity relation and higher-order functions, as well as the implementation of the resulting scheme in the CFLP(\mathcal{C})-system Toy (Arenas et al. 2007).

References

- APT, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B: Formal Models and Semantics. Elsevier and The MIT Press, 493–574.
- ARCELLI, F. AND FORMATO, F. 1999. Likelog: a logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied computing (SAC'99)*. ACM Press, New York, NY, USA, 260–267.
- ARCELLI FONTANA, F. 2002. Likelog for flexible query answering. *Soft Computing* 7, 107–114.
- ARENAS, P., FERNÁNDEZ, A. J., GIL, A., LÓPEZ-FRAGUAS, F. J., RODRÍGUEZ-ARCALEJO, M., AND SÁENZ-PÉREZ, F. 2007. *TOY*, a multiparadigm declarative language. version 2.3.1. R. Caballero and J. Sánchez (Eds.), Available at <http://toy.sourceforge.net>.

- BAADER, F. AND NIPKOW, T. 1998. *Term Rewriting and All That*. Cambridge University Press.
- BALDWIN, J. F., MARTIN, T., AND PILSWORTH, B. 1995. *Fril-Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons.
- BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 2001. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems* 3, 1 (January), 1–29.
- CABALLERO, R., RODRÍGUEZ-ARTELEJO, M., AND ROMERO-DÍAZ, C. A. 2008. Similarity-based reasoning in qualified logic programming. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming*. ACM, Valencia, Spain, 185–194.
- CABALLERO, R., RODRÍGUEZ-ARTELEJO, M., AND ROMERO-DÍAZ, C. A. 2009. Qualified computations in functional logic programming. In *Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. LNCS, vol. 5649. Springer-Verlag Berlin Heidelberg, Pasadena, CA, USA, 449–463.
- CAMPI, A., DAMIANI, E., GUINEA, S., MARRARA, S., PASI, G., AND SPOLETINI, P. 2009. A fuzzy extension of the XPath query language. *Journal of Intelligent Information Systems* 33, 3 (December), 285–305.
- DUBOIS, D. AND PRADE, H. 1980. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, NY, USA.
- FREUDER, E. C. AND WALLACE, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58, 1–3, 21–70.
- GEORGET, Y. AND CODOGNET, P. 1998. Compiling semiring-based constraints with CLP(FD,S). In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*. LNCS, vol. 1520. Springer-Verlag, 205–219.
- GUADARRAMA, S., MUÑOZ, S., AND VAUCHERET, C. 2004. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems* 144, 1, 127–150.
- HÁJEK, P. 1998. *Metamathematics of Fuzzy Logic*. Dordrecht: Kluwer.
- HÖHFELD, M. AND SMOLKA, G. 1988. Definite relations over constraint languages. Tech. Rep. LILOG Report 53, IBM Deutschland.
- ISHIZUKA, M. AND KANAI, N. 1985. Prolog-ELF incorporating fuzzy logic. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, A. K. Joshi, Ed. Morgan Kaufmann, Los Angeles, CA, USA, 701–703.
- JAFFAR, J. AND LASSEZ, J. L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87)*. ACM New York, NY, USA, Munich, West Germany, 111–119.
- JAFFAR, J., MAHER, M., MARRIOTT, K., AND STUCKEY, P. J. 1998. Semantics of constraints logic programs. *Journal of Logic Programming* 37, 1–3, 1–46.
- JULIÁN, R., MORENO, G., AND PENABAD, J. 2009. An improved reductant calculus using fuzzy partial evaluation techniques. *Fuzzy Sets and Systems* 160, 2, 162–181.
- JULIÁN-IRANZO, P., RUBIO, C., AND GALLARDO, J. 2009. Bousi~Prolog: a prolog extension language for flexible query answering. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 131–147.
- JULIÁN-IRANZO, P. AND RUBIO-MANZANO, C. 2009a. A declarative semantics for Bousi~Prolog. In *PPDP'09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*. ACM, Coimbra, Portugal, 149–160.
- JULIÁN-IRANZO, P. AND RUBIO-MANZANO, C. 2009b. A similarity-based WAM for Bousi~Prolog. In *Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN 2009)*. LNCS, vol. 5517. Springer Berlin / Heidelberg, Salamanca, Spain, 245–252.

- KIFER, M. AND SUBRAHMANIAN, V. S. 1992. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming* 12, 3&4, 335–367.
- LEE, R. C. T. 1972. Fuzzy logic and the resolution principle. *Journal of the Association for Computing Machinery (ACM)* 19, 1 (January), 109–119.
- LI, D. AND LIU, D. 1990. *A Fuzzy Prolog Database System*. John Wiley & Sons.
- LLOYD, J. W. 1987. *Foundations of Logic Programming, Second Edition*. Springer.
- LOIA, V., SENATORE, S., AND SESSA, M. I. 2004. Similarity-based SLD resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems* 144, 1, 151–171.
- MEDINA, J., OJEDA-ACIEGO, M., AND VOJTÁŠ, P. 2001a. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning (LPNMR’01)*, T. Eiter, W. Faber, and M. Truszczyński, Eds. LNAI, vol. 2173. Springer-Verlag, 351–364.
- MEDINA, J., OJEDA-ACIEGO, M., AND VOJTÁŠ, P. 2001b. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence (EPIA’01)*, P. Brazdil and A. Jorge, Eds. LNAI, vol. 2258. Springer-Verlag, 290–297.
- MEDINA, J., OJEDA-ACIEGO, M., AND VOJTÁŠ, P. 2004. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets and Systems* 146, 43–62.
- RIEZLER, S. 1998. Probabilistic constraint logic programming. Ph.D. thesis, Neuphilologischen Fakultät der Universität Tübingen.
- RODRÍGUEZ-ARTALEJO, M. AND ROMERO-DÍAZ, C. A. 2008. Quantitative logic programming revisited. In *Functional and Logic Programming (FLOPS’08)*, J. Garrigue and M. Hermenegildo, Eds. LNCS, vol. 4989. Springer-Verlag, Ise, Japan, 272–288.
- RODRÍGUEZ-ARTALEJO, M. AND ROMERO-DÍAZ, C. A. 2010a. A declarative semantics for CLP with qualification and proximity. *Theory and Practice of Logic Programming, 26th Int’l. Conference on Logic Programming (ICLP’10) Special Issue* 10, 4–6, 627–642.
- RODRÍGUEZ-ARTALEJO, M. AND ROMERO-DÍAZ, C. A. 2010b. Fixpoint & Proof-theoretic Semantics for CLP with Qualification and Proximity. Tech. Rep. SIC-1-10, Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain.
- SESSA, M. I. 2001. Translations and similarity-based logic programming. *Soft Computing* 5, 2.
- SESSA, M. I. 2002. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science* 275, 1-2, 389–426.
- VAN EMDEN, M. H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming* 3, 1, 37–53.
- VOJTÁŠ, P. 2001. Fuzzy logic programming. *Fuzzy Sets and Systems* 124, 361–370.
- ZADEH, L. A. 1965. Fuzzy sets. *Information and Control* 8, 3, 338–353.
- ZADEH, L. A. 1971. Similarity relations and fuzzy orderings. *Information Sciences* 3, 2, 177–200.